RICARDO TAVARES DE OLIVEIRA

REDUÇÕES DE PROBLEMAS EM GRAFOS COM SOLUÇÕES CONEXAS PARA (MAX)SAT E ADAPTAÇÃO DE UM RESOLVEDOR SAT E MAXSAT NÃO CLAUSAL PARA AS INSTÂNCIAS OBTIDAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Fabiano Silva

CURITIBA

2013

RICARDO TAVARES DE OLIVEIRA

REDUÇÕES DE PROBLEMAS EM GRAFOS COM SOLUÇÕES CONEXAS PARA (MAX)SAT E ADAPTAÇÃO DE UM RESOLVEDOR SAT E MAXSAT NÃO CLAUSAL PARA AS INSTÂNCIAS OBTIDAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Fabiano Silva

CURITIBA

2013

Oliveira, Ricardo Tavares de
Reduções de problemas em grafos com soluções conexas para (MAX)SAT e adaptação de um resolvedor SAT e MaxSAT não clausal para as instâncias obtidas / Ricardo Tavares de Oliveira. – Curitiba, 2013.
102 f. : il.; tab.

Dissertação (mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Fabiano Silva

1. Programação lógica. I. Silva, Fabiano. II. Título.

CDD 005.115

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Ricardo Tavares de Oliveira, avaliamos o trabalho intitulado, *"Reduções de problemas em grafos com soluções conexas para (max)sat e adaptação de um resolvedor sat e maxsat não clausal para as instâncias obtidas"*, cuja defesa foi realizada no dia 26 de fevereiro de 2013, às 13:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 26 de fevereiro de 2013.

Prof. Dr. Fabiano Silva
DINF/UFPR – Orientador

Prof. Dr. Cláudio Cesar de Sá
UDESC – Membro Externo

Prof. Dr. Renato Carmo
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço a meu orientador, Prof. Dr. Fabiano Silva, pelo compartilhamento aberto de sua sabedoria e pelas excelentes sugestões técnicas e científicas fornecidas durante as discussões particulares e em grupo, que foram crucial para a formação de um perfil pesquisador.

Agradeço em particular aos Profs. Drs. Marcos Castilho, Renato Carmo e André Guedes, pela disponibilidade e disposição de conhecer, discutir e opinar de maneira madura sobre este trabalho, tanto em discussões informais quanto nas apresentações do mesmo.

Agradeço a meus colegas de laboratório, em particular ao mestre Bruno César Ribas, pelo esforço aplicado em seu trabalho anterior e pelo fornecimento cordial do mesmo, utilizado como parte básica deste trabalho. Também agradeço aos demais membros do LIAMF, em particular os mestres Marcos Schreiner, Willian Zalewski e Razer Montaño, pela amizade e companhia diária.

Agradeço à comunidade brasileira de organizadores e participantes da Maratona de Programação, competição para a qual o treinamento auxiliou no desenvolvimento de raciocínio lógico, na aprendizagem de novas estruturas de dados e na capacidade de programação rápida, fatores estes que contribuíram para que o trabalho não tomasse mais tempo. Além de meus *coaches*, os já citados Bruno e André, agradeço em particular a Eduardo Augusto Ribas, pela enorme ajuda prestada durante os treinamentos. Também agradeço a meus colegas de equipe, Vinicius Ruoso, Flávio Zavan e Fernando Gielow, por tornarem as equipes da UFPR fortes.

Agradeço ao CNPq, pelo financiamento total deste trabalho.

Agradeço ao corpo docente do Departamento de Informática da UFPR como um todo, por conter professores que apresentaram de maneira excelente, ao longo de toda a minha graduação e meu mestrado, todo o conteúdo necessário para se formar um bom profissional da Computação. Pela boa dedicação, além dos já mencionados Fabiano, Renato, Marcos e André, pode-se citar o Prof. Dr. Roberto Hexel, a Profa. Dra. Carmem Hara, o Prof.

Dr. Elias Duarte Jr., e vários outros professores do departamento.

Agradeço à minha família, em particular aos meus pais, Hélio Almeida Oliveira e Roseli Tavares, pela excelente educação e criação, e por terem demostrado, durante toda a minha vida, total apoio a todos os meus trabalhos e esforços.

Agradeço aos meus colegas e amigos de turma, em particular a Paulo Ferreira, Grazielle Vernize, William Komura e Juan Rubial, além dos já citados Vinicius, Fernando e Eduardo. Tal grupo foi e é responsável por eternizar ótimos momentos durante e após o período do curso.

Agradeço ao Centro de Computação Científica e *Software* Livre (C3SL), onde estagiei por bastante tempo e fiz boas amizades, como as com Diego Pasqualin e Danilo Yorinori, entre várias outras.

Por fim, agradeço aos gatos da minha casa, Mitsi e Fritz, pela companhia calorosa em dias e noites que passei trabalhando neste documento.

SUMÁRIO

LISTA DE SIGLAS E ABREVIAÇÕES vii

LISTA DE FIGURAS viii

LISTA DE TABELAS ix

LISTA DE ALGORITMOS x

RESUMO xi

ABSTRACT xii

1 INTRODUÇÃO 1

2 DEFINIÇÕES INICIAIS 5

2.1 Teoria de Grafos 5

2.1.1 Definições 5

2.1.2 Descrição de problemas em grafos 8

2.2 Satisfabilidade Booleana 11

2.3 Satisfabilidade Booleana Máxima 13

2.4 Operador Escolhe-H 15

2.5 Considerações 17

3 REDUÇÕES DE PROBLEMAS DE GRAFOS PARA (MAX)SAT 18

3.1 Reduções conhecidas 18

3.1.1 Árvore de Steiner 19

3.1.2 Ciclo Hamiltoniano 21

3.1.3 Clique Máxima 22

3.2 Novas reduções 23

3.2.1 Verificação de Caminhos 24

3.2.2 Caminho Mínimo 26
3.2.3 Verificação de conexidade 27
3.2.4 Árvore de Steiner 29
3.2.5 Ciclo Hamiltoniano 31
3.2.6 Ciclo Hamiltoniano Mínimo 35
3.2.7 Clique 36
3.3 Considerações 38

4 MÉTODOS DE RESOLUÇÃO DE SATISFABILIDADE BOOLEANA E MÁXIMA 39

4.1 O problema de Satisfabilidade Booleana 39
4.1.1 O procedimento DPLL 39
4.1.2 Técnicas para melhoria do desempenho do DPLL 41
4.1.3 Satisfabilidade Não Clausal 46
4.2 O problema de Satisfabilidade Máxima 49
4.2.1 O procedimento DPLL como ramificação e poda 50
4.2.2 Fatores de desempenho do algoritmo de ramificação e poda 51
4.2.3 Técnicas e problemas relacionados 53
4.3 Considerações 57

5 MODIFICAÇÕES NO RESOLVEDOR LIAMFSAT 58

5.1 Modificações essenciais 58
5.1.1 Modificações no formato ISCAS 58
5.1.2 Modificações na função BCP() do LIAMFSAT 60
5.1.3 O LIAMFSAT como um resolvedor MaxSAT 60
5.2 O Módulo de Análise 61
5.3 O Módulo de Inferências 67
5.3.1 O Módulo de Inferências no Módulo de Análise 73
5.4 Considerações 74

6 RESULTADOS EXPERIMENTAIS **76**

6.1 Árvore de Steiner 80

6.2 Ciclo Hamiltoniano 85

6.2.1 Resolução do problema de decisão com SAT clausal 85

6.2.2 Resultados Obtidos 86

6.3 Ciclo Hamiltoniano Mínimo 89

6.4 Clique 91

6.5 Considerações 93

7 CONCLUSÃO E TRABALHOS FUTUROS **94**

BIBLIOGRAFIA **97**

LISTA DE SIGLAS E ABREVIAÇÕES

BCP	Boolean Constraint Propagation
BIT	Binary Indexed Tree (Árvore de Fenwick)
CI	Cota Inferior
CNF	Forma Normal Conjuntiva
CNFd	Redução apresentada na subseção 3.1.2
CNFj	Redução apresentada na subseção 3.1.1
CNFr	Conversão para CNF de fórmula gerada por uma redução da seção 3.2
CS	Cota Superior
DAG	Grafo Dirigido Acíclico
DPLLRP	DPLL com Ramificação e Poda
EPWMaxSAT	Satisfabilidade Máxima Estendida
LCOMP	Redução apresentada na subseção 3.1.3
MA	Módulo de Análise
MaxClique	Clique Máxima
MaxSAT	Satisfabilidade Booleana Máxima
MI	Módulo de Inferências
MLS	Resolvedor apresentado no capítulo 5
MLS$\bar{a}$	MLS sem aprendizado e retrocesso não cronológico
MLS$\bar{a}\bar{m}$	MLS sem aprendizado, retrocesso não cronológico e MA

MLS$\bar{m}$	MLS sem o Módulo de Análise
MLSj	MLS com a função de subjugação ativa
MLSj$\bar{m}$	MLS com a função de subjugação, mas sem o Módulo de Análise
MU	Mais de Um
NNF	Forma Normal Negada
PCB	Problema de Cobertura Binária
pql	preferência qualitativa de literais
PWMaxSAT	Satisfabilidade Máxima Ponderada e Parcial
SAT	Satisfabilidade Booleana
SAT-Pr	Satisfabilidade com Preferências
SUP	Técnica da seção 6.2.1
TA	Todos Acima
TLE	Tempo Limite Excedido
UC	Núcleo Insatisfatível
UIP	Unique Implication Point
UNR	Resolução de Vizinhança Unitária
WMaxSAT	Satisfabilidade Máxima Ponderada

LISTA DE FIGURAS

2.1 Representação de um grafo 5
2.2 Representação de um grafo ponderado 8
2.3 Representação de um grafo ponderado Hamiltoniano 10
2.4 Exemplo de uma instância de EPWMaxSAT 15

3.1 Uma árvore de um grafo destacada 19

4.1 Exemplo da construção do digrafo de implicações 44
4.2 Formato ISCAS da fórmula $\neg(x_1 \wedge x_2) \vee ((x_2 \vee \neg x_3) \wedge \neg(x_1 \vee x_3 \vee \neg x_4))$. 47

5.1 O formato ISCAS alterado para representar instâncias MaxSAT 60
5.2 Diagrama de fluxo de dados entre o LIAMFSAT e o Módulo de Análise . . 62
5.3 Estrutura de busca e união utilizada para detecção de ciclos. 63
5.4 Estrutura de busca e união atualizada se a aresta $e1$ é adicionada 64
5.5 Exemplo de um subgrafo $G_c(A)$ com $n = 3$ árvores. 66
5.6 Diagrama de fluxo de dados entre o LIAMFSAT e o Módulo de Inferências 68
5.7 *SetTrie* armazenando os conjuntos $\{0, 3, 6, 9\}, \{0, 13, 11, 3, 14\}$ e $\{6, 9\}$. . . . 70
5.8 Indicação da *cache* de cada nó de uma *SetTrie* para a valoração parcial $A = \{0, 11, 13, 14\}$ 71
5.9 Vetor L indicando listas ligadas com referências às arestas de mesmo rótulo. 72
5.10 Diagrama de fluxo de dados completo 74

6.1 Esquema da codificação em CNF de um somador de n entradas [38]. 78

LISTA DE TABELAS

6.1 Tamanhos das fórmulas obtidas pelas reduções com instâncias das categorias B e I080 da SteinLib 81
6.2 Resultados experimentais com instâncias das categorias B e I080 da SteinLib 82
6.3 Tamanhos das fórmulas obtidas pelas reduções com instâncias aleatórias . 83
6.4 Resultados experimentais com instâncias aleatórias 83
6.5 Tamanhos das fórmulas obtidas pelas reduções com instâncias criadas "à mão" do *Benchmark* do ASSAT 86
6.6 Resultados experimentais com instâncias criadas "à mão" do *Benchmark* do ASSAT 86
6.7 Tamanhos das fórmulas obtidas a partir de grafos completos 87
6.8 Resultados experimentais com instâncias com grafos completos 87
6.9 Tamanhos das fórmulas obtidas a partir de grafos completos aleatórios para Ciclo Hamiltoniano Mínimo 89
6.10 Resultados experimentais com grafos completos aleatórios para Ciclo Hamiltoniano Mínimo 90
6.11 Tamanhos das fórmulas obtidas das reduções para (Max)SAT com grafos esparsos 91
6.12 Resultados obtidos com instâncias contendo grafos esparsos 92

LISTA DE ALGORITMOS

1 DPLL 40

2 BCP() em fórmulas CNF 41

3 DPLL com Ramificação e Poda (DPLLRP) para MaxSAT 50

4 Algoritmo GSAT utilizado para cálculo da cota superior inicial 51

RESUMO

São apresentadas neste trabalho reduções de problemas de grafos para SAT ou MaxSAT. Reduções dos problemas da Árvore de Steiner, Ciclo Hamiltoniano e Ciclo Hamiltoniano Mínimo são apresentadas, assim como uma redução de Clique para SAT (embora uma redução mais simples de MaxClique para MaxSAT seja conhecida). Todas essas reduções são lineares ou quadráticas no tamanho do grafo dado.

Essas reduções usam um operador de cardinalidade que não está entre os operadores tradicionais da lógica proposicional. Entretanto, este operador pode ser convertido para uma fórmula em CNF em tempo linear, de forma que é possível utilizar um resolvedor SAT no estado-da-arte para resolver as instâncias geradas.

Também é válido tentar resolver essas instâncias com um resolvedor SAT não clausal que suporte o operador. Desta forma, o operador pode ser utilizado diretamente, sem a necessidade de sua conversão para uma fórmula em CNF.

Assim, este trabalho também apresenta uma versão modificada de um resolvedor SAT não clausal que reconhece o operador e resolve as instâncias de SAT e MaxSAT não clausal obtidas pelas reduções apresentadas.

Além disso, este resolvedor é auxiliado por dois novos módulos. Um destes módulos é capaz de antecipar os retrocessos do resolvedor através da análise de sua valoração parcial com o grafo original. O outro, mais genérico, é capaz de armazenar as cláusulas aprendidas durante o processo e inferir valores verdade de acordo com a valoração parcial.

As reduções apresentadas são comparadas com outras reduções publicadas anteriormente para os mesmos problemas, tanto no tamanho das fórmulas geradas quanto no desempenho de um resolvedor SAT as resolvendo. Além disso, o desempenho do resolvedor SAT não clausal modificado e seus módulos também é testado.

ABSTRACT

It's presented in this work some new reductions from problems in graphs to SAT or MaxSAT. Reductions from Steiner Tree, Hamiltonian Cycle and Minimum Hamiltonian Cycle are shown, as well a reduction from Clique to SAT (although there's a simpler known reduction from MaxClique to MaxSAT). All these reductions are linear or quadratic in the size of the given graph.

These reductions use a cardinality operator that is not among the traditional ones from the propositional logic. However, this operator can be translated to a CNF formula in linear time, so one can use a state-of-art clausal SAT solver to solve the generated instances.

It's also worth trying to solve these instances with a non-clausal SAT solver that supports the operator. By doing this, one can use the operator directly, without converting it to a CNF formula.

Thus, this work also presents a modified version of a non-clausal SAT solver that can handle this operator and solve the non-clausal SAT and MaxSAT instances obtained from the new reductions.

Also, this new solver is helped by two new modules. One of these modules is able to anticipate the solver's backtracks by analyzing its current partial assignment on the original graph. The other, more generic one, is able to store the clauses learnt during the process and to infer truth values according to the partial assignment.

The new reductions are compared against some other known reductions for the same problems, both in the size of the generated formulae and in the performance of a SAT solver working on them. Also, the performance of the modified non-clausal SAT solver and its new modules is tested.

CAPÍTULO 1

INTRODUÇÃO

O estudo da redutibilidade entre problemas computacionais desperta interesse tanto na teoria quanto em aplicações práticas. Na teoria, a redutibilidade auxilia na classificação de problemas computacionais de acordo com sua complexidade teórica. Isto permite, por exemplo, a construção das classes de problemas, como P e NP.

O interesse prático da redutibilidade consiste principalmente no fato de que um problema pode ser resolvido com algoritmos e técnicas utilizados para a resolução de outros problemas. Isto é particularmente verdade em reduções de problemas para o problema de Satisfabilidade Booleana (SAT, que consiste em decidir se há uma valoração de variáveis booleanas que satisfaz uma dada fórmula), o de Satisfabilidade Booleana Máxima (MaxSAT, que consiste em encontrar uma valoração que maximiza o número de cláusulas satisfeitas de uma fórmula na Forma Normal Conjuntiva (CNF)), ou uma de suas generalizações, como o problema MaxSAT Parcial (que consiste em encontrar uma valoração que satisfaz um dado conjunto de cláusulas, e maximiza o número de cláusulas satisfeitas entre as demais), ou MaxSAT Parcial Ponderado (onde há um peso associado a cada cláusula).

Existem muitos resolvedores SAT e MaxSAT eficientes atualmente, principalmente os *clausais*, que resolvem instâncias na Forma Normal Conjuntiva [11, 24, 1]. Esses resolvedores, que utilizam um algoritmo potencialmente exponencial como base, são capazes de resolver instâncias consideradas grandes pela comunidade em tempo satisfatório. Dessa forma, é possível acreditar que algumas instâncias reduzidas de problemas "intratáveis" para SAT ou MaxSAT podem ser resolvidas em tempo satisfatório por um resolvedor. Por isso, há o interesse em reduções de problemas "intratáveis" para SAT ou MaxSAT, e em suas resoluções com um resolvedor no estado-da-arte.

Dentre os problemas "intratáveis" destacam-se aqueles provenientes da Teoria de Gra-

fos. Como grafos têm grande poder representativo, os grafos são comumente utilizados para a modelagem de problemas práticos. Os problemas da Teoria de Grafos têm então uma grande importância na resolução de problemas relevantes.

A redução de um problema em grafos para SAT comumente consiste na associação de uma variável booleana a cada vértice e/ou aresta do grafo, e na construção de uma fórmula que codifica um conjunto de dadas restrições. A fórmula construída é satisfeita se e somente se uma dada restrição é cumprida. Restrições podem ser informalmente exemplificadas como *"vértices não vizinhos não devem estar na solução"*, *"as cores de vértices vizinhos devem ser diferentes"* ou *"a solução deve ser um subgrafo conexo"*.

O problema da Clique Máxima, por exemplo, que consiste em encontrar o maior subgrafo completo de um grafo G dado, pode ser reduzido para MaxSAT Parcial associando-se uma variável booleana x_i a cada vértice $v_i \in V(G)$ do grafo G, cujo valor verdade é verdadeiro se e somente se x_i pertence a uma clique máxima de G. Constrói-se então uma fórmula que é satisfatível se e somente se *vértices não vizinhos não estão na solução*. Esta restrição pode ser codificada pela conjunção de $(\neg x_i \wedge \neg x_j)$, para todo $x_i, x_j : \{x_i, x_j\} \notin E(G)$. Uma outra restrição, *o número de vértices deve ser maximizado*, também é codificada na parte de otimização da fórmula [24].

Enquanto a codificação de algumas restrições é trivial, não é intuitivo codificar outras, como *"a solução deve ser um subgrafo conexo"*. Embora foram publicadas algumas reduções de problemas que necessitam desta restrição [30, 5, 25, 24], não existe na literatura, ao melhor de nosso conhecimento, uma codificação fácil e intuitiva desta restrição em particular. Este é um dos principais motivadores deste trabalho.

Um dos objetivos deste trabalho é apresentar reduções eficientes de problemas em grafos para SAT e MaxSAT, particularmente aqueles cuja solução deve ser um subgrafo conexo. Destacam-se os problemas da Árvore de Steiner, do Ciclo Hamiltoniano, do Ciclo Hamiltoniano Mínimo e de Clique. Todas as reduções apresentadas neste trabalho são lineares (Clique) ou quadráticas (Árvore de Steiner e Ciclo Hamiltoniano (Mínimo)).

As reduções apresentadas geram fórmulas que não estão em CNF, e que utilizam um operador lógico diferente daqueles definidos pela lógica proposicional clássica (negação,

disjunção e conjunção). Esse operador, batizado de *Escolhe-H*, é um operador de cardinalidade utilizado para codificar principalmente relações entre os graus dos vértices do subgrafo buscado.

Este operador pode ser convertido para uma fórmula em CNF em tempo linear, como apresentado neste trabalho. Assim, é possível utilizar um resolvedor SAT ou MaxSAT específico para formulas em CNF para resolver as instâncias. Como os resolvedores clausais no estado-da-arte são aceitavelmente eficientes, esta abordagem para a resolução dessas instâncias é interessante.

Entretanto, também é possível adaptar um resolvedor SAT que resolve instâncias que não estão em CNF (dito *não clausal*) para suportar o novo operador proposto e resolver as instâncias geradas pelas reduções apresentadas.

A adaptação de um resolvedor não clausal para essas instâncias é interessante por permitir o uso do operador de forma direta, sem a conversão para operadores clássicos. A conversão do operador para uma fórmula CNF, embora linear, adiciona um conjunto considerável de novas variáveis e cláusulas à fórmula. Através do uso de um resolvedor não clausal adaptado para suportar o operador, o tamanho da fórmula não é incrementado, e o resolvedor então trabalha com uma representação menor da instância.

A contribuição deste trabalho consiste então em dois itens distintos:

- Apresentar novas reduções para SAT ou MaxSAT de problemas em grafos cuja solução deve ser um subgrafo conexo de um dado grafo, em particular o problema da Árvore de Steiner, o de Ciclo Hamiltoniano, o de Ciclo Hamiltoniano Mínimo e o de Clique;

- Adaptar um resolvedor SAT não clausal para suportar o operador utilizado nas reduções, resolver as instâncias de SAT e MaxSAT geradas pelas mesmas e comparar seu desempenho com resolvedores clausais.

O LIAMFSAT [41] foi utilizado como resolvedor não clausal base neste trabalho.

Sabendo-se que as instâncias foram obtidas por reduções de problemas em grafos, foi embutido um módulo no LIAMFSAT que conhece o grafo original da instância e

otimiza o processo de busca do mesmo, através de cortes antecipados no espaço de busca. Esses cortes são idênticos aos que um procedimento de ramificação e poda que trabalha diretamente no grafo original poderia fazer.

Além disso, um segundo módulo, capaz de armazenar as cláusulas aprendidas durante o processo e realizar inferências sobre elas, foi embutido no LIAMFSAT. Embora não esteja diretamente relacionado com as reduções apresentadas, este módulo implementa características inicialmente ausentes do resolvedor. Além disso, o segundo módulo também pode auxiliar o processo do primeiro.

O trabalho está estruturado da seguinte maneira:

O capítulo 2 apresenta as definições necessárias para a compreensão do restante do texto. O capítulo 3 apresenta reduções de problemas em grafos para SAT e MaxSAT, tanto algumas publicadas anteriormente quanto as novas, apresentadas neste trabalho. O capítulo 4 apresenta uma revisão bibliográfica dos algoritmos utilizados pelos resolvedores SAT e MaxSAT no estado-da-arte. O capítulo 5 apresenta as adaptações realizadas no LIAMFSAT para resolver as instâncias geradas pelas novas reduções.

O capítulo 6 apresenta uma comparação experimental de todas as reduções apresentadas, tanto em tamanho quanto em desempenho de resolução. Por fim, o capítulo 7 apresenta uma conclusão e alguns possíveis trabalhos futuros.

CAPÍTULO 2

DEFINIÇÕES INICIAIS

Esse capítulo apresenta definições inicias necessárias para a compreensão do trabalho. São definidos os problemas da Teoria de Grafos utilizados, além dos problemas relacionados com satisfabilidade booleana.

2.1 Teoria de Grafos

Este trabalho assume que o leitor tem alguma familiaridade com os conceitos da área da Teoria de Grafos e Complexidade Computacional. Assim, as definições necessárias para sua compreensão são dadas sucintamente.

2.1.1 Definições

Um *grafo* G é um par $G = (V, E)$ de conjuntos, onde $E \subseteq \binom{V}{2}$ é um conjunto de subconjuntos de tamanho 2 de V. Os elementos de V são ditos *vértices* de G, enquanto os elementos de E são ditas *arestas* de G. Os conjuntos V e E também podem ser denotados por $V(G)$ e $E(G)$ respectivamente, indicando que ambos pertencem ao grafo G.

A figura 2.1 representa o grafo $G = (V, E)$, onde $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, e $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}, \{v_5, v_6\}, \{v_6, v_8\}, \{v_8, v_7\}, \{v_5, v_7\}\}$.

Um *subgrafo* $G' = (V', E')$ de G é um grafo em que $V' \subseteq V$ e $E' \subseteq E$, isto é, um grafo

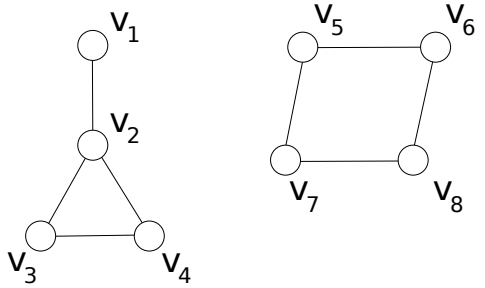

Figura 2.1: Representação de um grafo

cujos (cujas) vértices (arestas) consistem em um subconjunto dos (das) vértices (arestas) de G. Note que G' deve ser um grafo, isto é, a propriedade $E' \subseteq \binom{V'}{2}$ deve ser mantida. $G' = (\{v_1, v_2, v_5\}, \{\{v_1, v_2\}\})$ é um subgrafo do grafo da figura 2.1.

A *vizinhança* (ou *estrela*) de um vértice $v \in V(G)$ no grafo G, denotada por $N(v)$, é o conjunto de arestas de G que contém v, isto é, $N(v) = \{e_i \in E(G) : v \in e_i\}$. O *grau* de um vértice $v \in V(G)$ no grafo G, denotado por $d(v)$, é o número de arestas em sua vizinhança, isto é, $d(v) = |N(v)|$. No grafo da figura 2.1, a vizinhança do vértice v_7, que tem grau 2, é dada pelo conjunto $\{\{v_7, v_8\}, \{v_7, v_5\}\}$.

Um *caminho* P de tamanho k entre dois vértices v_a e v_b de $V(G)$ é um conjunto ordenado $P = (e_1, e_2, ..., e_k)$ de arestas de G tal que:

- $v_a \in e_1$ e nenhuma aresta em $P \setminus \{e_1\}$ contém v_a;
- $v_b \in e_k$ e nenhuma aresta em $P \setminus \{e_k\}$ contém v_b;
- $|e_i \cap e_{i+1}| = 1$ para $1 \leq i < k$, isto é, existe apenas e exatamente um vértice v_i em comum às arestas adjacentes de P, e nenhuma aresta em $P \setminus \{e_i, e_{i+1}\}$ contém o vértice v_i.

No grafo da figura 2.1, $P = (\{v_1, v_2\}, \{v_2, v_4\})$ é um caminho entre os vértices v_1 e v_4.

Um caminho P de tamanho k, quando conveniente, pode ser reescrito como a sequência $P = (v_1, v_2, ..., v_{k+1})$ de $k+1$ vértices, na qual $v_1 = v_a$, $v_{k+1} = v_b$ e $v_i \in e_i \cap e_{i+1}$, isto é, os vértices v_i são os vértices que pertencem a arestas consecutivas de P. Ambas as notações são utilizadas nesse trabalho, dependendo de seu contexto.

Um caminho P é dito *vazio* se não contém arestas. Um caminho vazio entre os vértices v_a e v_b ocorre apenas quando $v_a = v_b$.

Um *ciclo* C de tamanho k é uma sequência de arestas $C = (..., e_1, e_2, ..., e_k, e_1, ...)$, $k \geq 3$, onde $|e_i \cap e_{(i \bmod k)+1}| = 1$ para $1 \leq i \leq k$, isto é, existe apenas e exatamente um vértice v_i em comum a arestas adjacentes na sequência (note que a última aresta é adjacente à primeira). v_i também não deve ocorrer em nenhuma outra aresta de C, exceto as duas adjacentes correspondentes.

Existem dois ciclos no grafo da figura 2.1. Um deles é dado pela sequência cíclica $C = (\{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_2\})$.

Um grafo $G = (V, E)$ é dito *conexo* se existe em G algum caminho entre cada par de vértices $v_a, v_b \in V$. Um grafo G é dito *acíclico* se não contém ciclos. Por fim, G é uma *árvore* se é simultaneamente conexo e acíclico. Se G é uma árvore, então G necessariamente tem $|E| = |V| - 1$ arestas [14].

No grafo da figura 2.1, o subgrafo $G' = (V', E')$ composto por $V' = \{v_1, v_2, v_3, v_4\}$ e $E' = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}\}$ é uma árvore.

Uma *componente conexa* de G é um subgrafo G' de G conexo maximal, isto é, não existe um subgrafo conexo G'' de G de forma que $V(G'') = V(G') \cup \{v_a\}$, para algum $v_a \in V(G') \backslash V(G)$, ou que $E(G'') = E(G') \cup \{e_i\}$, para algum $e_i \in E(G') \backslash E(G)$. Note que, se G é conexo, então G contém apenas uma componente conexa, igual ao próprio grafo G. Note também que não existe um caminho em G entre o par de vértices $v_a, v_b \in V(G)$ se (e somente se) v_a e v_b pertencem a componentes conexas de G distintas. Dois vértices estão *conectados* em G se pertencem à mesma componente conexa de G.

O grafo da figura 2.1 contém exatamente duas componentes conexas. Uma delas contém os vértices v_1, v_2, v_3 e v_4, enquanto a outra contém os demais vértices do grafo.

Cada componente conexa de um grafo acíclico G consiste em uma árvore. Um grafo acíclico também é dito uma *floresta*. Este termo é mais utilizado para enfatizar sua composição em árvores, ao invés de sua ausência de ciclos.

Um grafo $G = (V, E)$ é dito *completo* se seu conjunto de arestas contém todas as combinações possíveis de pares de seus vértices, isto é, se $E = \binom{V}{2}$. Uma *clique* de G de tamanho k é um subgrafo completo de G que contém k vértices.

No grafo da figura 2.1, o subgrafo que contém os vértices v_2, v_3 e v_4 e as arestas $\{v_2, v_3\}, \{v_3, v_4\}$ e $\{v_2, v_4\}$ é uma clique de tamanho $k = 3$.

O subgrafo de G *induzido* por um conjunto de arestas $E' \subseteq E$ consiste no grafo G' que contém as arestas de E' e os vértices da união das arestas em E', isto é, $G' = (V', E')$, tal que $V' = \cup_{(e_i \in E')} e_i$.

A clique exemplificada acima é o subgrafo induzido pelo conjunto de arestas $E' =$

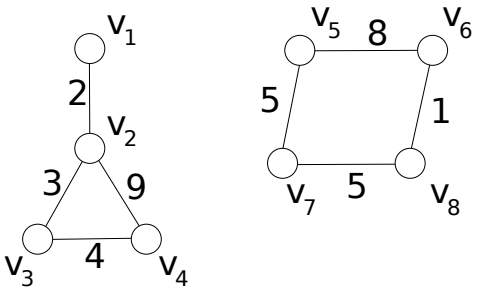

Figura 2.2: Representação de um grafo ponderado

$\{\{v_2, v_3\}, \{v_3, v_4\}, \{v_2, v_4\}\}$.

A *união* dos grafos $G_1, G_2, ..., G_k$ é dada por $(V(G_1) \cup ... \cup V(G_k), E(G_1) \cup ... \cup E(G_k))$.

Um grafo G é dito *ponderado* se está associado a uma função $w : E(G) \rightarrow \mathbb{R}^+$ que associa, a cada aresta e_i de G, um *peso* real positivo $w(e_i)$. O *peso* (ou *custo*) de um caminho, ciclo ou subgrafo é dado pela soma dos pesos das arestas que o compõe.

A figura 2.2 representa um grafo ponderado cujos pesos das arestas $\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}$, $\{v_3, v_4\}, \{v_5, v_6\}, \{v_6, v_8\}, \{v_8, v_7\}, \{v_7, v_5\}$ são $2, 3, 9, 4, 8, 1, 5$ e 5, respectivamente.

O *complemento* de um grafo $G = (V, E)$ é dado pelo grafo $\bar{G} = (V, E')$, onde $\{v_i, v_j\} \in E'$ se e somente se $\{v_i, v_j\} \notin E$, para $v_i, v_j \in V, v_i \neq v_j$.

Finalmente, um *digrafo*, grafo *dirigido* ou grafo *direcionado* é um par $G = (V, E')$ onde $E' \subseteq V^2$. Os elementos de E' são ditos *arcos* de G. Informalmente, um digrafo é um grafo em cada aresta assume uma orientação.

2.1.2 Descrição de problemas em grafos

Como descrito no capítulo 1, os problemas de interesse neste trabalho são os da Árvore de Steiner, Ciclo Hamiltoniano, Ciclo Hamiltoniano Mínimo, e Clique, por requererem que a solução seja um subgrafo conexo e por não existirem reduções simples conhecidas destes problemas para os de satisfabilidade. Estes problemas são descritos nesta seção.

Alguns outros problemas também são apresentados, pois seus conceitos são utilizados nas reduções dos problemas de interesse. Os problemas utilizados nesse trabalho são:

- **Verificação de Caminhos:** Dado um grafo G e dois vértices v_a, v_b distintos de $V(G)$, o problema consiste em decidir se há um caminho entre v_a e v_b em G. Esse

problema pode ser resolvido em tempo linear, por exemplo, com o algoritmo de Busca em Profundidade [14], e tem diversas aplicações em diversas áreas da Computação.

No grafo da figura 2.1, há um caminho entre os vértices v_1 e v_3, mas não há um caminho entre os v_2 e v_6.

- **Caminho Mínimo:** Este problema de otimização consiste em, dado um grafo ponderado G e um par v_a, v_b de vértices de G, encontrar um caminho entre v_a e v_b de custo mínimo. O problema pode ser resolvido em $O((|V| + |E|) \log(|V|))$ com o algoritmo de Dijkstra [18]. Tem aplicação clara no planejamento de rotas, por exemplo.

  No grafo da figura 2.2, o caminho mínimo entre os vértices v_1 e v_4 contém, além destes, os vértices v_2 e v_3, e tem peso 9.

- **Verificação de Conexidade:** Dado um grafo G e um subconjunto $S \subseteq V(G)$ de seus vértices, o problema de verificação de conexidade consiste em decidir se todos os vértices de S estão conectados em G, isto é, se existe um caminho em G entre todo par de vértices de S. Esse problema também pode ser resolvido em tempo linear com uma Busca em Profundidade, e sua resolução pode ser utilizada como subrotina para algoritmos para outros problemas, como o da Árvore de Steiner.

  No grafo da figura 2.1, os vértices $S = \{v_1, v_2, v_4\}$ estão conectados, mas os vértices $S = \{v_1, v_4, v_5, v_6\}$ não estão.

- **Árvore de Steiner:** O problema da árvore de Steiner consiste em, dado um grafo ponderado G e um subconjunto $S \subseteq V(G)$, encontrar um subgrafo de G de peso mínimo que conecta os vértices de S. Como o peso de cada aresta de G é positivo, a solução desde problema é acíclica e, logo, uma árvore. Este problema está na classe NP-Difícil [29], e pode ser resolvido em tempo $O^*(c^{|S|})^1, c > 2$ com a técnica de Programação Dinâmica [22]. Este problema é uma generalização do problema da

[1] $f(n) = O^*(g(n))$ se $f = O(h(n) \times g(n))$, onde $h(n)$ é um polinômio.

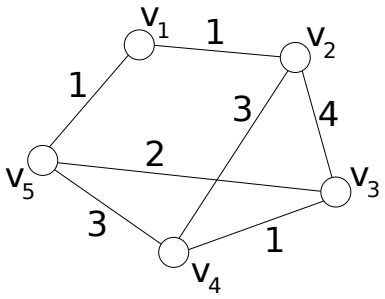

Figura 2.3: Representação de um grafo ponderado Hamiltoniano

Árvore Geradora Mínima, onde $S = V(G)$, que pode ser resolvido em tempo polinomial, por exemplo, pelo algoritmo de Kruskal [32]. Esse problema tem aplicações principalmente na área de Geometria Computacional e Projeto de Circuítos [47].

Na figura 2.2, a árvore de Steiner para $S = \{v_1, v_3, v_4\}$ consiste no subgrafo $G' = (V', E')$ tal que $V' = \{v_1, v_2, v_3, v_4\}$ e $E' = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}\}$, e tem peso 9.

- **Ciclo Hamiltoniano**: Dado um grafo G, este problema consiste em verificar se existe um ciclo em G que contém todos os seus vértices. Se G possui tal ciclo, G é dito Hamiltoniano.

  Este problema também pertence à classe NP-Completo [29], e pode ser resolvido, também através da técnica de Programação Dinâmica, em tempo $O(2^{|V|}|V|^2)$ [9].

  O grafo da figura 2.1 não é Hamiltoniano, enquanto o da figura 2.3 é. Há dois ciclos hamiltonianos neste grafo.

- **Ciclo Hamiltoniano Mínimo** (ou *Caixeiro Viajante*): Dado um grafo ponderado G, este problema de otimização consiste em encontrar um ciclo de custo mínimo em G que contém todos os vértices em $V(G)$. Este problema pertence à classe NP-Difícil [29], e pode ser resolvido por programação dinâmica com a mesma complexidade computacional apresentada para o problema do Ciclo Hamiltoniano [9]. Uma de suas aplicações é a modelagem do problema Euclidiano do Caixeiro Viajante.

  O Ciclo Hamiltoniano Mínimo do grafo da figura 2.3 tem custo 8.

- **Clique:** Este problema de decisão consiste em, dado um grafo G e um inteiro $k \in \mathbb{N}, k > 1$, decidir se G contém uma clique de tamanho k. O problema está na classe NP-completo [29], pode ser resolvido por técnicas de retrocesso (*backtracking*), e tem aplicações na análise de redes sociais, por exemplo.

  Como exemplificado, o grafo da figura 2.1 contém uma clique de tamanho 3.

Todos os problemas citados são reduzidos para os problemas de Satisfabilidade ou Satisfabilidade Máxima no capítulo 3. Assim, é necessário definir tais problemas.

2.2 Satisfabilidade Booleana

Algumas definições provenientes da Teoria da Lógica Proposicional Clássica são necessárias para que o problema de Satisfabilidade seja definido.

Uma variável booleana (ou proposição) é uma variável cujo domínio é o conjunto de *valores verdade* $\mathbb{B} = \{0, 1\}$. O valor verdade 0 pode ser interpretado como falso, enquanto 1 pode ser interpretado como verdadeiro.

Sintaticamente, *operadores* (ou *conectivos lógicos*) são símbolos utilizados para relacionar as variáveis de um dado conjunto $\mathscr{X}$ entre si. Na *lógica proposicional clássica* são definidos três operadores básicos: a *negação* ($\neg$), a *disjunção* ($\vee$) e a *conjunção* ($\wedge$). A negação tem aridade unitária, enquanto os demais têm aridade variável. Uma *fórmula* é uma variável de $\mathscr{X}$ ou a relação entre outras fórmulas (ditas *subfórmulas*) através de um operador. Por exemplo, seja $\mathscr{X} = \{x_1, x_2, x_3\}$. $f = (x_1 \vee \neg x_2) \wedge (x_3)$ é uma fórmula, pois utiliza o operador $\wedge$ para relacionar as subfórmulas $(x_1 \vee \neg x_2)$ e x_3. De forma análoga, $(x_1 \vee \neg x_2)$ também é uma fórmula . Por fim, x_3 também é uma fórmula por ser uma variável de $\mathscr{X}$.

Semanticamente, um operador de aridade k é uma função $o : \mathbb{B}^k \to \mathbb{B}$ que associa, a uma sequência de k subfórmulas com valores verdade definidos, um valor verdade. O valor verdade de uma subfórmula é definido então como sendo o valor verdade associado ao operador utilizado para conectar suas subfórmulas, ou o valor de uma variável em $\mathscr{X}$ caso tal subfórmula seja composta apenas por ela. Na lógica proposicional define-se a

seguinte semântica para seus operadores:

- Negação ($\neg f$, "não"): Assume o inverso do valor verdade de f, isto é, $\neg f = 1 - f$.
- Disjunção ($f_1 \vee f_2 \vee ... \vee f_k$, "ou"): Assume 1 se existe algum j em $[1..k]$ tal que f_j tem valor verdade 1, e 0 caso contrário. Desta forma, $(f_1 \vee f_2 \vee ... \vee f_k) = max\{f_1, f_2, ..., f_k\}$.
- Conjunção ($f_1 \wedge f_2 \wedge ... \wedge f_k$, "e"): Assume 1 se f_j tem valor verdade 1 para todo j em $[1..k]$, e 0 caso contrário. Desta forma, $(f_1 \wedge f_2 \wedge ... \wedge f_k) = min\{f_1, f_2, ..., f_k\}$.

Uma *valoração* sobre um conjunto $\mathscr{X}$ de variáveis é uma função $A : \mathscr{X}' \to \mathbb{B}$, $\mathscr{X}' \subseteq \mathscr{X}$, que associa a cada variável de um subconjunto de variáveis, um valor verdade. Uma valoração é *completa* se associa um valor verdade a todas as variáveis, isto é, $\mathscr{X}' = \mathscr{X}$. Caso contrário, a valoração é *parcial*.

Um *literal* é uma variável x_i (dito *literal positivo*) ou sua negação $\neg x_i$ (dito *literal negativo*). $var(l)$ denota a variável do literal l, isto é, $var(x_i) = var(\neg x_i) = x_i$. $sig(l)$, por sua vez, denota o valor verdade de $var(l)$ que torna o literal l verdadeiro, isto é, $sig(x_i) = 1$ e $sig(\neg x_i) = 0$. Uma *cláusula* é uma disjunção de literais, que também pode ser denotada como o conjunto dos literais presentes na mesma. Uma fórmula é *clausal* ou está na *Forma Normal Conjuntiva* (*Conjunctive Normal Form* ou CNF) se é uma conjunção de cláusulas, e também pode ser denotada como um conjunto de cláusulas. Como exemplo, $f = (x_1 \vee \neg x_2) \wedge (x_3)$ está em CNF, e pode ser denotado por $f = \{\{x_1, \neg x_2\}, \{x_3\}\}$. Uma fórmula que não está no formato CNF é dita *não clausal*.

Um *modelo* para uma fórmula é uma valoração completa que torna 1 o valor verdade da fórmula. Como exemplo, a valoração $\{(x_1, 0), (x_2, 0), (x_3, 1)\}$ é um modelo para f. Em fórmulas em CNF, uma valoração *satisfaz* uma cláusula se torna seu valor verdade igual a 1, e *satisfaz* uma fórmula se satisfaz todas as suas cláusulas.

Uma valoração ou um modelo também pode ser representado como um conjunto A de literais, onde $x_i \in A$ se e somente se o valor verdade associado a x_i é 1, e $\neg x_i \in A$ se e somente se o valor associado é 0. Como exemplo, o modelo apresentado para f pode ser denotado por $\{\neg x_1, \neg x_2, x_3\}$.

O problema de Satisfabilidade Booleana (SAT) consiste em, dada uma fórmula f, verificar se f é *satisfatível*, isto é, se existe um modelo para f. O problema SAT foi o primeiro problema cuja pertinência à classe de problemas NP-Completo foi provada [13].

Além de servir como uma base teórica no estudo de redutibilidade entre problemas, o problema SAT tem aplicações práticas, por exemplo, na área de verificação de circuitos [41]. O capítulo 4 apresenta algoritmos e suas otimizações capazes de resolver este problema.

2.3 Satisfabilidade Booleana Máxima

Verificar se uma dada fórmula é satisfatível não é suficiente para algumas aplicações reais, sendo necessário quantificar a satisfabilidade da mesma.

O problema de *Satisfabilidade Máxima* (MaxSAT) consiste em, dada uma fórmula em CNF, encontrar uma valoração que maximize o número de cláusulas satisfeitas. Como exemplo, seja f a fórmula $f = (x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$, a qual não é satisfatível, pois não existe valoração que satisfaça suas três cláusulas. Entretanto, a valoração $A = \{\neg x_1, \neg x_2\}$ satisfaz duas de suas cláusulas, e é uma resposta para o problema MaxSAT desta instância. Note que, enquanto SAT é um problema de decisão, MaxSAT é um problema de otimização.

MaxSAT é uma especialização de outro problema, o de Satisfabilidade Máxima *Ponderada* (WMaxSAT). O problema consiste em, dada uma fórmula f em CNF e uma função $W : f \rightarrow \mathbb{R}^+$ de ponderação de cláusulas, encontrar uma valoração cuja soma dos pesos das cláusulas satisfeitas é máxima. Para facilitar a leitura deste trabalho, uma cláusula $C \in f$ e seu peso $W(C)$ pode ser denotado pelo par $(C, W(C))$.

O problema WMaxSAT, por sua vez, é uma especialização do problema de Satisfabilidade Máxima Ponderada *e Parcial* (PWMaxSAT). Seja $W : f \rightarrow \mathbb{R}^+ \cup \{\top\}$ uma função que associa, a cada cláusula de uma fórmula em CNF f, um número real positivo ou o símbolo $\top$. Cláusulas cujo peso é o simbolo $\top$ são ditas *hard*, enquanto as demais são ditas *soft*. O problema PWMaxSAT consiste em encontrar uma valoração que satisfaça todas as cláusulas *hard* e, simultaneamente, maximize a soma dos pesos das cláusulas *soft*

satisfeitas. Como exemplo, seja f a fórmula $f = (x_1, 13) \wedge (\neg x_1 \vee \neg x_2, 3) \wedge (\neg x_1 \vee x_2, \top)$. A valoração $A = \{x_1, x_2\}$ satisfaz todas as cláusulas *hard* e satisfaz cláusulas *soft* cuja soma dos pesos é 13, máxima para esta instância. Pode-se denotar por f_h a conjunção das cláusulas *hard* de uma fórmula f, e por f_s a conjunção do restante das cláusulas. Note que $f = f_h \wedge f_s$.

O problema PWMaxSAT está na classe de problemas NP-Difícil, uma vez que o problema SAT pode ser descrito como uma especialização de PWMaxSAT, cujas entradas consistem em fórmulas em que todas as suas cláusulas são classificadas como *hard*.

Nesse trabalho, considera-se o problema de Satisfabilidade Máxima *Estendida*, denominada aqui por EPWMaxSAT.

Neste trabalho, uma fórmula f é dita *estendida* se, além dos três operadores clássicos definidos na lógica proposicional, f pode conter outros operadores na forma $\mathbb{B}^k \to \mathbb{B}$. A fórmula f não precisa estar descrita em CNF para ser considerada estendida.

O problema EPWMaxSAT consiste em, dada uma fórmula *hard* f_h, estendida ou não, dado um conjunto de fórmulas *soft* f_s estendidas ou não, e uma função $W : f_s \to \mathbb{R}^+$ de ponderação das fórmulas *soft*, encontrar uma valoração $A : \mathscr{X} \to \mathbb{B}$ que satisfaz a fórmula f_h e cuja soma dos pesos das fórmulas *soft* satisfeitas por A é máxima.

Tal problema é idêntico a PWMaxSAT, exceto pela remoção da restrição de que as fórmulas de entrada devam estar em formato clausal e devam conter apenas operadores clássicos da lógica proposicional.

A figura 2.4 exemplifica uma instância deste problema, que consiste em uma fórmula *hard* não clausal, além de duas fórmulas *soft*, com pesos 42 e 19.

Esse problema também está na classe de problemas NP-Difícil, uma vez que SAT também pode ser descrito como especialização sua.

Neste texto, o termo “MaxSAT” pode ser utilizado para se referir ao problema EPWMaxSAT ou aos seus problemas correlatos apresentados. Como MaxSAT, WMaxSAT e PWMaxSAT são especializações de EPWMaxSAT, tal convenção não invalida as definições dadas acima. O termo “fórmula” também é utilizado para se referir, de fato, a uma fórmula estendida ou não clausal. A classificação da fórmula à qual o termo se refere,

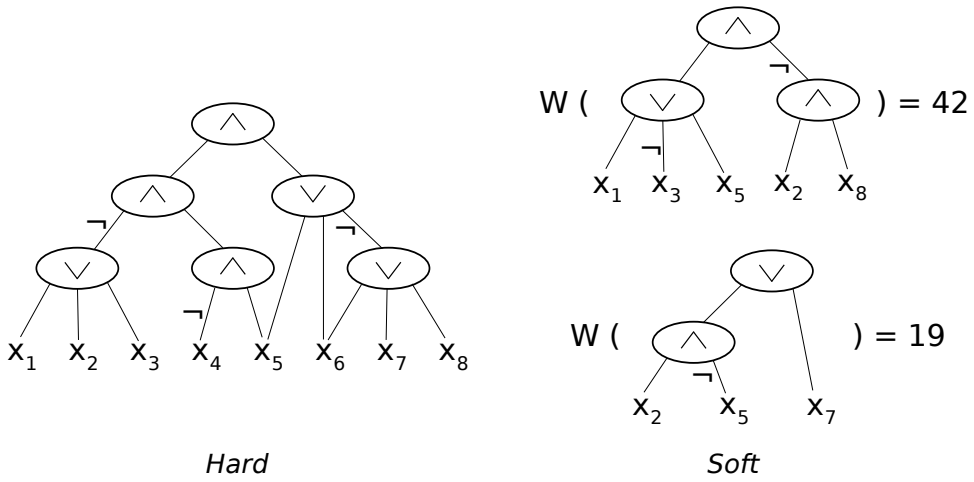

Figura 2.4: Exemplo de uma instância de EPWMaxSAT

quando necessária, é feita pelo contexto do texto.

O capítulo 4 apresenta métodos conhecidos para a resolução destes problemas.

2.4 Operador Escolhe-H

Para facilitar as reduções dos problemas em grafos apresentados aos problemas SAT e MaxSAT, um operador lógico, distinto dos operadores clássicos descritos pela lógica proposicional, é utilizado.

Este é um *operador de cardinalidade*. Operadores de cardinalidade, informalmente, assumem um valor de acordo com a contagem da quantidade de valores verdadeiros (ou falsos) em todos ou parte de seus argumentos.

Seja $H \subseteq \{0, ..., k\}$ um conjunto finito de números naturais menores ou iguais a um dado $k \in \mathbb{N}$. O operador *Escolhe-H* (C_H), de aridade k, é definido pela a seguinte semântica:

$$C_H(f_1, f_2, ..., f_k) = \begin{cases} 1, \mathsf{se}(\sum_{i=1}^{k} f_i) \in H \\ 0, \mathsf{caso\ contrário} \end{cases}$$

Assim, esse operador assume o valor verdade 1 se e somente se o número de suas subfórmulas com valor verdade 1 está presente no conjunto H. O operador $C_{\{0,2\}}(f_1, ..., f_k)$, por exemplo, assume o valor 1 se e somente se nenhuma ou exatamente duas de suas subfórmulas são consideradas verdadeiras.

Para facilitar a leitura do texto, opcionalmente denota-se o operador $C_{\{n\}}()$ por $C_n()$, quando $H = \{n\}$ for um conjunto unitário. Além disso, a notação $C_H(S)$, quando S é um conjunto qualquer, denota o operador $C_H(s_1, s_2, ..., s_{|S|})$, onde $\{s_1, ..., s_{|S|}\} = S$.

A extensão de MaxSAT definida na seção anterior permite a utilização do operador Escolhe-H nas fórmulas de entrada do problema descrito. Este operador é componente essencial em todas as reduções apresentadas no capítulo 3, pois pode ser utilizado principalmente para modelar restrições sobre o grau de vértices em grafos.

É válido notar também que todos os operadores definidos na lógica proposicional podem ser convertidos para um operador Escolhe-H em tempo linear, através da observação das seguintes propriedades:

- $(f_1 \wedge ... \wedge f_k) = C_k(f_1, ..., f_k)$

  Uma conjunção de k subfórmulas tem valor verdadeiro se e somente se todas as suas k subfórmulas também o têm;

- $(f_1 \vee ... \vee f_k) = C_{\{1,...,k\}}(f_1, ..., f_k)$

  Uma disjunção de k subfórmulas tem valor verdadeiro se uma ou mais de suas k subfórmulas também o têm;

- $\neg f = C_0(f)$

  A negação de uma fórmula f tem valor verdadeiro se e somente se nenhuma fórmula do conjunto unitário $\{f\}$ tem tal valor.

Além disso, a seguinte propriedade também é valida:

- $\neg C_H(f_1, ..., f_k) = C_{\{0,...,k\} \setminus H}(f_1, ..., f_k)$.

Note que a propriedade acima pode ser utilizada para remover o operador de negação de operadores Escolhe-H. Como toda fórmula que contém operadores da lógica proposicional pode ser convertida para fórmulas contendo operadores Escolhe-H, é possível fazer com que o operador $\neg$ seja utilizado sobre variáveis apenas.

Este operador pode ser convertido para uma fórmula em CNF em tempo linear em seu tamanho, como apresentado no capítulo 6.

Outros operadores de cardinalidade já foram publicados e observados anteriormente na comunidade [2, 35, 7]. O capítulo 6 apresenta uma breve descrição de cada um deles e desmotiva seu uso, particularmente durante sua conversão para CNF.

2.5 Considerações

Este capítulo apresenta as definições iniciais necessárias para o entendimento do restante do trabalho. As definições apresentadas neste capítulo são utilizadas em todo o trabalho, incluindo os capítulos 3 e 4. Esses capítulos apresentam as novas reduções de problemas em grafos para SAT e uma revisão bibliográfica de resolvedores SAT e MaxSAT, respectivamente.

CAPÍTULO 3

REDUÇÕES DE PROBLEMAS DE GRAFOS PARA (MAX)SAT

Este capítulo apresenta reduções de problemas da área de Teoria de Grafos para SAT ou MaxSAT. Uma *redução* de um problema P_1 a um problema P_2 consiste em uma função de associação das instâncias de P_1 às instâncias de P_2 de forma que a resposta de uma instância de P_1 é idêntica à respostosta de sua instância de P_2 associada.

O objetivo do capítulo é apresentar reduções dos problemas da Árvore de Steiner, do Ciclo Hamiltoniano, do Ciclo Hamiltoniano Mínimo e de Clique, por serem problemas cujas soluções devem ser um subgrafo conexo e por não haver reduções simples conhecidas dos mesmos para SAT e/ou MaxSAT.

O capítulo está dividido em duas seções. A primeira apresenta algumas reduções publicadas anteriormente, enquanto a segunda apresenta as novas reduções, descritas neste trabalho.

3.1 Reduções conhecidas

Esta seção apresenta reduções publicadas anteriormente dos problemas de interesse neste trabalho. São descritas reduções dos problemas da Árvore de Steiner, do Ciclo Hamiltoniano e de Clique Máxima. Uma redução do problema do Ciclo Hamiltoniano Mínimo é facilmente obtida pela redução apresentada do problema de Ciclo Hamiltoniano. Além disso, o problema de Clique pode ser trivialmente resolvido com as soluções do problema de Clique Máxima, como descrito na subseção 3.1.3.

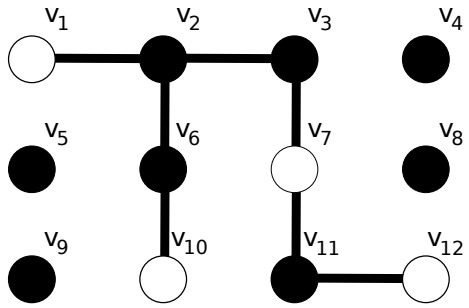

Figura 3.1: Uma árvore de um grafo destacada

3.1.1 Árvore de Steiner

É possível reduzir polinomialmente o problema da Árvore de Steiner para MaxSAT clausal se a árvore buscada for descrita como uma sequência de arcos resultante da execução de uma busca em profundidade na mesma [30].

Seja $G = (V, E), S \subseteq V$ e $w : E \to \mathbb{R}^+$ uma instância do problema da Árvore de Steiner.

Primeiramente, o grafo G é redefinido em um digrafo $G' = (V, E')$, onde cada aresta $\{v_i, v_j\} \in E(G)$ corresponde a um par de arcos $(v_i, v_j), (v_j, v_i) \in E'$.

Seja T a Árvore de Steiner de G procurada, representada por uma sequência de arcos $T = ((v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), ..., (v_{i_2}, v_{i_1}))$, onde $(v_{i_j}, v_{i_{j+1}})$ é um arco do digrafo G' e $v_{i_{j+1}}$ sucede v_{i_j} em uma busca em profundidade na árvore. Note que o primeiro e último vértices da sequência são idênticos. Na árvore exemplificada na figura 3.1, onde S é dado por $\{v_1, v_7, v_{10}, v_{12}\}$, a sequência pode ser descrita por $T = ((v_1, v_2), (v_2, v_3), (v_3, v_7),$
$(v_7, v_{11}), (v_{11}, v_{12}), (v_{12}, v_{11}), (v_{11}, v_7), (v_7, v_3), (v_3, v_2), (v_2, v_6), (v_6, v_{10}), (v_{10}, v_6), (v_6, v_2), (v_2, v_1))$.

A descrição completa da redução não foi explícita em sua publicação [30]. Assim, a redução seguinte foi desenvolvida com base nas ideias publicadas. O seu tamanho é compatível com o tamanho fornecido pelos autores da mesma.

A cada arco (v_i, v_j) do digrafo e a cada $t \in \mathbb{N}, 1 \leq t \leq |T|$, associa-se uma variável booleana $x_{i,j,t}$ que tem valor verdade 1 em uma valoração ótima se e somente se o arco (v_i, v_j) está na $t-$ésima posição de T.

Para que tal associação seja possível, é necessário definir o tamanho da sequência T anteriormente à redução. Tal valor pode ser fixado em $|T| = |E'| = 2 \times |E|$ se for

permitido que a sequência contenha um mesmo arco mais de uma vez. Toda sequência T de tamanho menor que $|E'|$ pode ser aumentada sem perda de corretude se os arcos (v_{i_1}, v_{i_2}) e (v_{i_2}, v_{i_1}) forem anexados ao final da sequência quantas vezes forem necessárias.

Assim, $|E'| \times |T| = O(|E|^2)$ variáveis são utilizadas nas fórmulas geradas.

A fórmula *hard* é constituída de uma conjunção de subfórmulas que representam as seguintes restrições:

- Todo vértice de S deve pertencer a algum arco de T.

  Tal restrição pode ser garantida, para cada $v_i \in S$, com a disjunção dos literais $x_{i,j,t}$, para todo $(v_i, v_j) \in E'$ e para todo $1 \leq t \leq |T|$.

  São criadas $O(|S|)$ cláusulas neste formato;

- Se há um arco (v_i, v_j) na posição t, então há um arco (v_j, v_k) na posição $t' = (t \ mod \ |T|) + 1$.

  Esta restrição pode ser representada, para cada arco $(v_i, v_j) \in E'$ e cada $1 \leq t \leq |T|$, pela cláusula $(\neg x_{i,j,t} \vee x_{j,k_1,t'} \vee x_{j,k_2,t'} \vee ...)$, onde as arestas $\{v_j, v_{k_i}\}$ formam a vizinhança de v_j em G.

  Devido a operação de resto de divisão (*mod*) utilizada, esta restrição garante que o primeiro e último vértices da sequência são idênticos.

  São criadas $O(|E| \times |T|)$ cláusulas neste formato;

- Se o arco (v_i, v_j) está em alguma posição de T, então o arco (v_j, v_i) também está em alguma posição de T.

  Tal restrição pode ser garantida, para cada arco $(v_i, v_j) \in E'$ e cada $1 \leq t \leq |T|$, com a cláusula $(\neg x_{i,j,t} \vee x_{j,i,1} \vee x_{j,i,2} \vee ... \vee x_{j,i,|T|})$.

  São criadas também $O(|E| \times |T|)$ cláusulas neste formato;

- Há no máximo um arco em cada posição de T. Esta restrição pode ser representada, para cada $1 \leq t \leq |T|$ e para cada par $(v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2})$ de arcos do digrafo, pela cláusula $(\neg x_{i_1,j_1,t} \vee \neg x_{i_2,j_2,t})$.

São criadas $O(|T| \times |E|^2)$ cláusulas neste formato.

As cláusulas *soft*, por sua vez, consistem em uma conversão para CNF das fórmulas $\neg(x_{i,j,1} \vee x_{i,j,2} \vee ... \vee x_{i,j,|T|})$ com peso $w(e_i)$, para cada arco $(v_i, v_j) \in E'$.

Essas fórmulas indicam que, se algum arco da aresta e_i está presente na sequência T, então o peso de e_i está presente na árvore de Steiner.

Devido à necessidade da presença de ambos os arcos de uma aresta, a valoração encontrada pelo resolvedor MaxSAT após esta redução, aplicada às cláusulas *soft*, terá o dobro do peso da árvore descrita de fato.

Embora correta, tal redução gera uma fórmula de tamanho considerado inviável na prática para instâncias grandes do problema, pois utiliza $O(|E|^2)$ variáveis e, apenas para a fórmula *hard*, $O(|S| + |E| \times |T| + |E| \times |T| + |T| \times |E|^2) = O(|E|^3)$ cláusulas.

Além de reduções exatas, também existem reduções aproximadas do problema da Árvore de Steiner para MaxSAT clausal. Dentre elas, destaca-se uma redução que, para cada vértice $s_i \in S$, limita a solução a um número fixo de caminhos mínimos entre s_i e os demais vértices [5].

Para que esta redução gere soluções exatas, é necessário enumerar todos os caminhos entre os vértices de S. Por isso, esta redução foi descartada nesse trabalho.

3.1.2 Ciclo Hamiltoniano

Semelhantemente à redução apresentada do problema da Árvore de Steiner, é possível reduzir o problema de Ciclo Hamiltoniano para SAT clausal se o ciclo buscado for descrito como uma sequência de $|V|$ *vértices* distintos $T = (v_1, v_2, ..., v_{|V|})$ onde $\{v_j, v_{(j \ mod \ |V|)+1}\} \in E(G)$, para todo $1 \leq j \leq |V|$ [25].

Associa-se a cada $v_i \in V(G)$ e cada $1 \leq j \leq |V|$ uma variável booleana $x_{i,j}$, que assume verdadeiro na valoração buscada se e somente se o vértice v_i está na $j-$ésima posição de T.

É possível então construir uma fórmula em CNF que é satisfatível se e somente se o grafo G é hamiltoniano. Tal fórmula consiste na conjunção de subfórmulas que representam as seguintes restrições:

- Existe ao menos um vértice em cada posição de T. Isto pode ser representado pela cláusula $(x_{1,j} \vee x_{2,j} \vee ... \vee x_{|V|,j})$, para cada posição $1 \leq j \leq |V|$.

  São criadas $|V|$ cláusulas neste formato;

- Cada vértice está presente em, no máximo, uma posição de T. Isto pode ser representado, para cada $v_i \in V(G)$ e para cada $1 \leq j_1 < j_2 \leq |V|$, pela cláusula $(\neg x_{i,j_1} \vee \neg x_{i,j_2})$.

  São criadas $\frac{|V|^3 - |V|^2}{2}$ cláusulas neste formato;

- Existe uma aresta contendo cada par de vértices adjacentes de T. Isto pode ser representado, para cada posição $1 \leq j \leq |V|$ e para cada aresta $\{v_i, v_k\} \notin E(G)$, pela cláusula $(\neg x_{i,j} \vee \neg x_{k,(j \ mod \ |V|)+1})$.

  São criadas $|V||E(\bar{G})|$ cláusulas neste formato.

Outros conjuntos de restrições podem ser utilizados nesta redução, mas as citadas acima são suficientes para a corretude da mesma [25].

A fórmula obtida através dessa redução utiliza $|V|^2$ variáveis e tem $|V| + \frac{|V|^3 - |V|^2}{2} + |V||E(\bar{G})| = O(|V|^3)$ cláusulas.

Esta redução pode ser facilmente adaptada para reduzir o problema do Ciclo Hamiltoniano Máximo para MaxSAT. Basta utilizar como fórmula construída nesta redução como *hard*, e criar subfórmulas que representam a ausência de cada aresta, convertidas para CNF, como fórmulas *soft*.

3.1.3 Clique Máxima

O problema de Clique pode ser resolvido através da análise da solução do problema de Clique Máxima.

O problema de Clique Máxima consiste em, dado um grafo $G = (V, E)$, encontrar o maior natural k tal que G contém uma clique de tamanho k ou, equivalentemente, tal que a resposta do problema Clique é positiva.

Há uma redução simples do problema de Clique Máxima para MaxSAT clausal, linear no tamanho do complemento do grafo G [24].

Associa-se, para cada vértice $v_i \in V(G)$, uma variável booleana x_i, que tem valor verdade 1 no modelo ótimo se e somente se tal vértice pertence à clique procurada.

A fórmula *hard* f_h consiste no conjunto de cláusulas $f_h = \{(\neg x_i \vee \neg x_j) : \{v_i, v_j\} \notin E(G)\}$. Essa fórmula indica que dois vértices não devem estar simultaneamente na solução buscada caso não estejam conectados por uma aresta em G.

As fórmulas *soft* são dadas pelas cláusulas unitárias (x_i) com peso 1, para cada $v_i \in V(G)$. Assim, a solução maximiza o número de vértices presentes na clique.

A redução apresentada é simples e as cláusulas de f_h tem tamanho 2 cada, apenas. Entretanto, essa redução é linear no tamanho do complemento de G, e não no tamanho do próprio grafo G. Assim, a fórmula gerada pode ser inviavelmente grande para grafos esparsos.

Uma instância G, k do problema de Clique pode ser resolvida através da resolução do problema de Clique Máxima com o grafo G, e da comparação de seu resultado com o número k: uma clique de tamanho k existe em G se e somente se o tamanho de uma clique máxima de G é maior ou igual a k.

3.2 Novas reduções

Esta seção apresenta as reduções propostas neste texto dos problemas de interesse para SAT e MaxSAT, que compõe uma das duas principais contribuições deste trabalho.

Como já descrito, o objetivo é apresentar reduções dos problemas da Árvore de Steiner, do Ciclo Hamiltoniano, do Ciclo Hamiltoniano Mínimo e de Clique. Entretanto, reduções de todos os problemas apresentados na seção 2.1.2 são descritas. Embora alguns destes problemas tenham solução polinomial conhecida, tais reduções são utilizadas como parte das reduções dos problemas de interesse, e portanto são relevantes para a compreensão das mesmas.

3.2.1 Verificação de Caminhos

Seja $G = (V, E)$ um grafo e $v_a, v_b \in V$ um par de seus vértices. Seja $f_p(G, v_a, v_b)$ uma fórmula booleana satisfatível se e somente se existe um caminho entre os vértices v_a e v_b em G. Além disso, qualquer modelo A para esta fórmula descreve um subgrafo $G_p(A)$ de G que contém um caminho entre os vértices dados. Esta fórmula pode ser definida da seguinte maneira:

- Associa-se, a cada $e_i \in E$, uma variável booleana x_i. Sendo $\mathscr{X}$ o conjunto de variáveis $\mathscr{X} = \{x_i : e_i \in E\}$, a aresta e_i está no grafo $G_p(A)$ se e somente se $A(x_i) = 1$, para qualquer valoração $A : \mathscr{X} \to \mathbb{B}$. O subgrafo $G_p(A)$ pode ser descrito então como o subgrafo de G induzido pelo conjunto de arestas $\{e_i : A(x_i) = 1\}$;

- Seja $\mathscr{N}(v_i) = \{x_j \in \mathscr{X} : e_j \in N(v_i)\}$ o conjunto de variáveis associadas à vizinhança do vértice $v_i \in V$. O grau de ambos os vértices v_a e v_b no grafo $G_p(A)$ deve ser igual a 1. Para tal, adiciona-se à fórmula as restrições $C_1(\mathscr{N}(v_a))$ e $C_1(\mathscr{N}(v_b))$;

- Para cada outro vértice v_i em $V \setminus \{v_a, v_b\}$, seu grau no subgrafo deve ser igual a 0 ou 2. Isto pode ser feito adicionando-se à fórmula a restrição $C_{\{0,2\}}(\mathscr{N}(v_i))$, para todo $v_i \in V$ diferente de v_a e v_b.

A fórmula $f_p(G, v_a, v_b)$ é então definida como a conjunção das restrições citadas:

$$f_p(G, v_a, v_b) = C_1(\mathscr{N}(v_a)) \wedge C_1(\mathscr{N}(v_b)) \wedge \tag{3.1}$$
$$C_{\{0,2\}}(\mathscr{N}(v_1)) \wedge C_{\{0,2\}}(\mathscr{N}(v_2)) \wedge ... \wedge C_{\{0,2\}}(\mathscr{N}(v_m))$$

onde $\{v_1, v_2, ..., v_m\} = V \setminus \{v_a, v_b\}$.

Para provar que existe um caminho entre v_a e v_b em G se e somente se $f_p(G, v_a, v_b)$ é satisfatível, os dois seguintes teoremas são apresentados:

Teorema 1: Se existe um caminho $P = (e_1, ..., e_k)$ entre v_a e v_b em G, então existe um modelo $A : \mathscr{X} \to \mathbb{B}$ para $f_p(G, v_a, v_b)$ para o qual $G_p(A)$ contém P.

Prova: Seja A a valoração que associa 1 para e apenas para as variáveis associadas às arestas que formam P, isto é, $A(x_i) = 1$ sse $e_i \in P$. Claramente $G_p(A)$ contém apenas as arestas de P. Logo, basta provar que A satisfaz a fórmula $f_p(G, v_a, v_b)$.

A aresta e_1 é a única aresta em P que contém o vértice v_a, devido às propriedades de um caminho descritas no capítulo 2. Assim, o grau de v_a em $G_p(A)$ é 1, o que satisfaz a restrição $C_1(\mathscr{N}(v_a))$. Da mesma forma, e_k é a única aresta que contém o vértice v_b, fazendo com que o grau de v_b também seja 1, satisfazendo a restrição $C_1(\mathscr{N}(v_b))$.

Como P é um caminho, as arestas e_i e e_{i+1}, para cada $1 \leq i < k$, contém um vértice em comum v_i. Tal vértice é diferente de v_a e v_b, e as arestas e_i e e_{i+1} são as únicas em P que o contém. Logo, o grau de v_i em $G_p(A)$ é igual a 2, o que satisfaz a restrição $C_{\{0,2\}}(\mathscr{N}(v_i))$. Para todos os outros vértices v_j que não pertencem a nenhuma aresta de P, não existem arestas de $N(v_j)$ em $G_p(A)$. Isso também satisfaz a restrição $C_{\{0,2\}}(\mathscr{N}(v_j))$, pois o grau de v_j é 0 em $G_p(A)$.

Assim, A satisfaz todas as restrições da fórmula $f_p(G, v_a, v_b)$. □

Teorema 2: Seja A uma valoração completa sobre $\mathscr{X}$. Se A satisfaz $f_p(G, v_a, v_b)$, então existe um caminho entre v_a e v_b no subgrafo $G_p(A)$.

Prova: Seja A uma valoração completa sobre $\mathscr{X}$ que satisfaz $f_p(G, v_a, v_b)$. Suponha que $G_p(A)$ não contém um caminho entre v_a e v_b, de forma que v_a e v_b pertencem a diferentes componentes conexas de $G_p(A)$. Seja $G'_p(A, v_a)$ o subgrafo de $G_p(A)$ que consiste na componente conexa a qual v_a pertence.

Como A satisfaz a fórmula dada, o grau de v_a em $G_p(A)$ é 1, e como $G'_p(A, v_a)$ é a uma componente conexa (maximal) que contém tal vértice, seu grau em $G'_p(A, v_a)$ também é 1. Note também que o único vértice, além de v_a, com grau ímpar (1) em $G_p(A)$, é v_b, que não pertence a $G'_p(A, v_a)$. Todos os demais vértices em $V \setminus \{v_a, v_b\}$ têm grau par (0 ou 2) em $G_p(A)$ e $G'_p(A, v_a)$, pelas restrições da fórmula satisfeitas por A. Desta forma, há um número ímpar (apenas um, v_a) de vértices de grau ímpar em $G'_p(A, v_a)$.

Entretanto, todo grafo deve conter um número par de vértices de grau ímpar [14]. Assim, a conclusão é um absurdo, pois $G'_p(A, v_a)$ é um grafo com um número ímpar de tais vértices.

Dessa forma, é impossível existir uma componente conexa de $G_p(A)$ que contém v_a e não contém v_b. Logo, existe um caminho entre ambos os vértices em $G_p(A)$. □

É válido notar que existem $|\mathscr{X}| = |E|$ variáveis e $|V|$ restrições na fórmula, uma para cada vértice de V. A aridade da restrição para o vértice $v_i \in V$ é igual a $|N(v_i)|$, o grau do vértice no grafo original G. Assim, o número de elementos em toda a fórmula é limitado a $|V|$ mais a soma dos graus dos vértices em G, igual a $2|E|$ [14]. Logo, o tamanho da fórmula resultante é linear no tamanho do grafo dado.

3.2.2 Caminho Mínimo

O problema do caminho mínimo entre dois vértices $v_a, v_b \in V(G)$ de um grafo G pode ser reduzido para MaxSAT definindo-se uma fórmula *hard* que indique a necessidade da existência de um caminho entre o par de vértices, e um conjunto de fórmulas *soft* que indicam o requerimento de minimização do peso de tal caminho.

A fórmula $f_p(G, v_a, v_b)$, definida na seção anterior pela equação 3.1, pode ser utilizada como fórmula *hard* nessa redução. O conjunto de fórmulas *soft* pode ser dado por $f_s = \{(\neg x_i, w(e_i)) : e_i \in E(G)\}$, ou seja, o conjunto de cláusulas unitárias $\neg x_i$, com peso $w(e_i)$, para cada aresta $e_i \in E(G)$. Essas fórmulas indicam a necessidade de se maximizar a soma dos pesos das arestas ausentes do caminho buscado, minimizando assim o peso de tal caminho. Assim, a instância de MaxSAT gerada pela redução é dada por:

$$\begin{aligned} f_h &: f_p(G, v_a, v_b) \\ f_s &: \{(\neg x_i, w(e_i)) : e_i \in E(G)\} \end{aligned} \tag{3.2}$$

P é um caminho mínimo entre v_a e v_b no grafo dado se e somente se existe uma solução ótima A para a instância de MaxSAT dada tal que $G_p(A)$ contém exatamente o caminho P. Isso pode ser observado pelos seguintes teoremas:

Teorema 3: Se $A : \mathscr{X} \to \mathbb{B}$ é uma solução ótima para a instância de MaxSAT dada, então $G_p(A)$ contém exatamente um caminho mínimo entre v_a e v_b.

Prova: Seja $A : \mathscr{X} \to \mathbb{B}$ um modelo para $f_p(G, v_a, v_b)$ e seja m a soma dos pesos das fórmulas *soft* satisfeitas por A.

Como cada fórmula *soft* consiste apenas na cláusula $\neg x_i$ cujo peso é $w(e_i)$ para cada $e_i \in E$, m é igual à soma dos pesos das arestas não presentes em $G_p(A)$. Seja $M = \sum_{e_i \in E} w(e_i)$ a soma dos pesos de todas as arestas de G.

Como o problema MaxSAT consiste em maximizar m, a soma dos pesos das arestas em $G_p(A)$, $M - m$, é minimizada. Além disso, segundo o teorema 2, existe um caminho entre v_a e v_b em $G_p(A)$. Logo, $G_p(A)$ é um subgrafo que consiste em um caminho mínimo entre ambos os vértices.

Assim, se A é uma solução ótima para a instância dada, $G_p(A)$ contém apenas um caminho mínimo entre v_a e v_b. □

Teorema 4: Seja P um caminho mínimo entre os vértices v_a e v_b de um grafo G. Existe uma solução ótima para a instância MaxSAT gerada pela redução para a qual $G_p(A)$ contém exatamente as arestas de P.

Prova: Seja P um caminho mínimo entre o par de vértices dado. Considere a valoração A apresentada pelo Teorema 1. A satisfaz $f_p(G, v_a, v_b)$ e $G_p(A)$ contém apenas as arestas de P. Seja $w(P)$ o peso do caminho P. Como $A(x_i) = 1$ para cada $e_i \in P$, todas e apenas as fórmulas *soft* correspondentes a uma aresta de P não são satisfeitas por A. Assim, a soma dos pesos de todas as fórmulas *soft* satisfeitas (a função objetivo do problema MaxSAT) é igual a $M - w(P)$. Como $w(P)$ é minimo, tal valor é máximo. Assim, A é uma solução ótima para a instância gerada. □

Como $|f_p(G, v_a, v_b)|$ é linear no tamanho do grafo G como observado na seção 3.2.1, e como há apenas uma cláusula *soft* unitária para cada aresta do grafo, esta redução ainda é linear no tamanho do grafo dado.

3.2.3 Verificação de conexidade

O problema de verificação de conexidade pode ser reduzido para SAT construindo-se uma fórmula $f_c(G, S)$ que é satisfatível se e somente se todos os vértices em S pertencem à mesma componente conexa de G. Analogamente à fórmula $f_h(G, v_a, v_b)$, todo modelo

A para $f_c(G, S)$ descreve um subgrafo $G_c(A)$ de G no qual todos os vértices de S estão conectados. Esta fórmula pode ser definida da seguinte maneira:

- Seja $G_1, G_2, ..., G_{|S|-1}$ uma coleção de grafos. Se o conjunto de vértices de G for definido por $V(G) = \{v_1, v_2, ..., v_{|V|}\}$, o grafo $G_j (1 \leq j < |S|)$ pode ser definido como $G_j = (V_j, E_j)$, onde $V_j = \{v_{1,j}, v_{2,j}, ..., v_{|V|,j}\}$ e $E_j = \{\{v_{k,j}, v_{m,j}\} : \{v_k, v_m\} \in E(G)\}$. Se a aresta $\{v_k, v_m\} \in E(G)$ é denotada por e_i, a aresta $\{v_{k,j}, v_{m,j}\} \in E(G_j)$ é denotada por $e_{i,j}$. Informalmente, G_j é uma "réplica" do grafo G indexada por j, e $e_{i,j}$ representa a aresta $e_i \in E(G)$ no grafo G_j.

- Associa-se, para cada $e_{i,j} \in \cup_{j=1}^{|S|-1} E(G_j)$, uma variável booleana $x_{i,j}$. Seja $\mathscr{X}_j = \{x_{i,j} : e_{i,j} \in E(G_j)\}$ e $\mathscr{X} = \cup_{j=1}^{|S|-1} \mathscr{X}_j$.

  Considere um modelo $A : \mathscr{X} \to \mathbb{B}$ para $f_c(G, S)$. A_j denota a valoração restrita às variáveis de $\mathscr{X}_j$, isto é, $A_j = \{(x_{i,j}, A(x_{i,j})) : x_{i,j} \in \mathscr{X}_j\}$.

  Considere também, para cada $1 \leq j < |S|$, o grafo $G_{j_p}(A_j)$, que consiste no subgrafo de G induzido pelo conjunto de arestas $\{e_i \in E(G) : A_j(x_{i,j}) = 1\}$. O grafo $G_c(A)$ é definido então como a união de todos os grafos $G_{j_p}(A_j)$.

- Seja $S' = (s_1, .., s_{|S|})$ uma permutação qualquer do conjunto S. Para que os vértices S estejam conectados em G, deve haver um caminho entre cada par de elementos adjacentes em S'. Isto pode ser codificado com a restrição $f_p(G_j, s_{j,j}, s_{j+1,j})$ definida pela equação 3.1, para todo $1 \leq j < |S|$. Por simplicidade, tal restrição também pode ser escrita como $f_p(G_j, s_j, s_{j+1})$ neste texto.

A fórmula $f_c(G, S)$ é definida então como a conjunção destas restrições, isto é:

$$f_c(G, S) = f_p(G_1, s_1, s_2) \wedge f_p(G_2, s_2, s_3) \wedge ... \wedge f_p(G_{|S|-1}, s_{|S|-1}, s_{|S|}) \tag{3.3}$$

A fórmula $f_c(G, S)$ é satisfatível se e somente se todos os vértices de S estão conectados em G, como mostrado pelos seguintes teoremas:

Teorema 5: Se todos os vértices de S estão conectados em G, então existe um modelo $A : \mathscr{X} \to \mathbb{B}$ para $f_c(G, S)$ para o qual todos os vértices de S estão conectados em $G_c(A)$.

Prova: Como todos os vértices em S pertencem à mesma componente conexa de G, existe um caminho em G entre cada par de vértices adjacentes em S'. Seja P_j o caminho entre os vértices s_j e s_{j+1}. Como mostrado no teorema 1, existe um modelo $A_j : \mathscr{X}_j \to \mathbb{B}$ para a fórmula $f(G_j, s_j, s_{j+1})$, para todo $1 \leq j < |S|$. Além disso, $G_{j_p}(A_j)$ contém P_j.

Seja $A : \mathscr{X} \to \mathbb{B}$ a união das valorações A_j descritas acima. Como A_j satisfaz $f_p(G_j, s_j, s_{j+1})$, A satisfaz todas as restrições de $f_c(G, S)$, e logo é um modelo para a fórmula dada.

Ainda, como $G_c(A)$ é definido como a união de todos os $G_{j_p}(A_j)$, que contém o caminho P_j, $G_c(A)$ contém um caminho entre todos os vértices adjacentes de S'. Por transitividade, $G_c(A)$ contém um caminho entre todos os pares de vértices em S e, logo, contém uma componente conexa a qual todos os vértices de S pertencem. □

Teorema 6: Seja A uma valoração completa sobre $\mathscr{X}$. Se A satisfaz $f_c(G, S)$, então todos os vértices de S estão conectados em $G_c(A)$ (e, logo, em G).

Prova: Seja $A : \mathscr{X} \to \mathbb{B}$ um modelo para $f_c(G, S)$. Como A satisfaz a fórmula $f_p(G_j, s_j, s_{j+1})$ para todo $1 \leq j < |S|$, existe um caminho P_j entre os vértices s_j e s_{j+1} no grafo $G_{j_p}(A_j)$ como mostrado no teorema 2.

Como $G_{j_p}(A_j)$ é um subgrafo de $G_c(A)$, então existe um caminho entre s_j e s_{j+1} em $G_c(A)$ para todo $1 \leq j < |S|$, também. Por transitividade, existe um caminho entre todo par de vértices de S em $G_c(A)$. □

Há $|S| - 1$ restrições em $f_c(G, S)$, as quais têm $|V| + 2|E|$ elementos cada, como mostrado na seção 3.2.1. Logo, o tamanho total da instância, que tem $|E|(|S| - 1)$ variáveis, é igual a $(|V| + 2|E|)(|S| - 1)$, quadrático no tamanho do grafo original, no pior caso. Entretanto, é válido notar que $|S| \leq |V|$ e, assim, o tamanho da fórmula é menor que $(|V| + |E|)^2$.

3.2.4 Árvore de Steiner

O problema da árvore de Steiner pode ser reduzido para MaxSAT pela construção de uma fórmula *hard* que indica a necessidade de que todos os vértices de S devem estar conectados na solução, e fórmulas *soft* que indicam a minimização da soma dos pesos das

arestas presentes na solução buscada.

Como fórmula *hard*, pode-se usar a fórmula $f_c(G, S)$ como definida na seção anterior pela equação 3.3. Cada fórmula *soft* pode ser definida como $f_{s_i} = (\neg x_{i,1} \wedge ... \wedge \neg x_{i,|S|-1})$ cujo peso é igual a $w(e_i)$, para cada $e_i \in E(G)$. Assim, a instância para MaxSAT gerada é dada por:

$$f_h : f_C(G, S) \tag{3.4}$$
$$f_s : \{((\neg x_{i,1} \wedge ... \wedge \neg x_{i,|S|-1}), w(e_i)) : e_i \in E(G)\}$$

A corretude de tal redução é mostrada nos seguintes teoremas:

Teorema 7: Se $A : \mathscr{X} \rightarrow \mathbb{B}$ é uma solução para a instância MaxSAT obtida, então $G_c(A)$ é uma árvore ótima contendo todos os vértices de S.

Prova: Seja $A : \mathscr{X} \rightarrow \mathbb{B}$ uma solução para a instância gerada. Como A satisfaz $f_c(G, S)$, $G_c(A)$ é um subgrafo de G no qual todos os vértices de S estão conectados, como mostra o teorema 6. A fórmula $f_{s_i} = (\neg x_{i,1} \wedge ... \wedge \neg x_{i,|S|-1})$ tem valor verdade 1 se e somente se nenhuma variável em $\{x_{i,1}, ..., x_{i,|S|-1}\}$ é verdadeira. Dessa forma, f_{s_i} é verdadeira se e somente se a aresta e_i está ausente de todos os grafos $G_{j_p}(A_j)$ e, logo, se a aresta não pertence a $G_c(A)$.

Assim, a soma dos pesos das fórmulas *soft* satisfeitas por A é igual à soma dos pesos das arestas não presentes em $G_c(A)$. De forma análoga ao teorema 3, tal soma é máxima e, portanto, $G_c(A)$ consiste na componente conexa contendo os vértices de S que tem o menor peso possível, o que caracteriza uma árvore ótima. □

Teorema 8: Seja T uma árvore ótima de G que contém os vértices de S. Existe um modelo A para $f_c(G, S)$ que é solução para a instância gerada para o qual $G_c(A) = T$.

Prova: Seja $S' = (s_1, ..., s_{|S|})$ a permutação de S utilizada durante a construção da fórmula $f_c(G, S)$. Como T é um subgrafo conexo de G, existe um caminho P_j entre os vértices s_j e s_{j+1} em T, para todo $1 \leq j < |S|$. Seja $A_j : \mathscr{X}_j \rightarrow \mathbb{B}$ a valoração que associa o valor verdade 1 à variável $x_{i,j}$ se e somente se a aresta e_i pertence ao caminho P_j, e seja

$A : \mathscr{X} \to \mathbb{B}$ a união de todos os A_j.

Como T é uma árvore, o caminho P_j entre s_j e s_{j+1} é único. Dessa forma, toda aresta de T pertence a pelo menos um caminho P_j, pois, se não pertencesse, haveria um outro caminho entre seus dois vértices no subgrafo. Logo, T pode ser dada como a união de todos P_j, $1 \leq j < |S|$. Assim, $G_c(A)$ contém apenas a árvore T, isto é, $G_c(A) = T$.

Além disso, a valoração A satisfaz $f_c(G, S)$, de forma análoga ao mostrado no teorema 5.

Por fim, como a fórmula f_{s_i} tem valor verdade 1 se e somente se a aresta e_i não está presente em $G_c(A)$, e como a soma dos pesos das arestas de T é mínima, a soma dos pesos das arestas ausentes de $G_c(A)$ é máxima, de forma análoga ao mostrado no teorema 4. Dessa forma, A é uma solução ótima da instância MaxSAT gerada. □

O número de elementos de $f_c(G, S)$ é igual a $(|V| + 2|E|)(|S| - 1)$, como apresentado na seção 3.2.3. Além disso, existem $|E|$ fórmulas em f_s com $|S| - 1$ elementos cada. Logo, a soma do tamanho de todas as fórmulas, *hard* e *soft*, é igual a $(|V| + 3|E|)(|S| - 1)$. Assim, essa redução é quadrática no tamanho do grafo original, no pior caso.

3.2.5 Ciclo Hamiltoniano

O problema de Ciclo Hamiltoniano pode ser reduzido para SAT através da construção é uma fórmula $f_H(G)$ que é satisfatível se e somente se G é Hamiltoniano. De forma análoga a outras reduções citadas, todo modelo A para $f_H(G)$ descreve um subgrafo $G_H(A)$ de G que consiste em um ciclo Hamiltoniano.

A fórmula $f_H(G)$ pode ser construída através das seguintes restrições:

- Todos os vértices da solução devem estar conexos. Isto pode ser codificado com a restrição $f_c(G, V)$, definida pela equação 3.3;
- Todos os vértices devem ter grau igual a 2 no ciclo buscado. Para cada aresta $e_i \in E(G)$, considere a fórmula

$$f_{e_i} = (x_{i,1} \vee x_{i,2} \vee ... \vee x_{i,|V|-1}) \tag{3.5}$$

A fórmula f_{e_i} assume valor verdade 1 se e somente se a aresta e_i está presente em $G_H(A)$. A restrição pode ser então representada pela fórmula $\wedge_{v_i \in V} C_2(\{f_{e_j} : e_j \in N(v_i)\})$.

A fórmula é então dada pela conjunção das restrições:

$$f_H(G) = f_c(G, V) \wedge \wedge_{v_i \in V} C_2(\{f_{e_j} : e_j \in N(v_i)\}) \tag{3.6}$$

A fórmula $f_H(G)$ é satisfatível se e somente se G é Hamiltoniano. A corretude dessa redução é mostrada pelos teoremas 9 e 11. O teorema 10 é utilizado como base durante a argumentação para o teorema 11. Entretanto, ele também poderia ser utilizado para justificar uma redução do problema do *Caminho* Hamiltoniano para SAT, não coberto neste trabalho devido à sua semelhança ao problema do Ciclo Hamiltoniano.

Teorema 9: Se G é um grafo Hamiltoniano, então existe um modelo A para $f_H(G)$, cujo subgrafo $G_H(A)$ é Hamiltoniano.

Prova: Seja G um grafo Hamiltoniano, e G' o subgrafo de G que consiste em um ciclo Hamiltoniano.

Seja $V' = (v_1, v_2, ..., v_{|V|})$ a permutação dos elementos de V utilizada durante a criação de $f_c(G, V)$, e seja $C' = (..., v_{c_1}, v_{c_2}, ..., v_{c_{|V|}}, v_{c_1}, ...)$ a sequência formada pela ordenação do ciclo no qual G' consiste em algum sentido.

Como G' é conexo e contém todos os vértices de V, existe em G' um caminho entre os vértices v_i e v_{i+1}, para todo $1 \leq i < |V|$. Em particular, existe o caminho $P'_i = (v_i = v_{c_j}, v_{c_{j+1}}, ..., v_{c_{j+|P'_i|-1}} = v_{i+1})$ que visita os vértices de v_i a v_{i+1} na ordem de C'. Dessa forma, toda aresta de G' está presente em pelo menos um caminho P'_i.

Considere as valorações A_j onde $A_j(x_{i,j}) = 1$ se e somente se $e_i \in P'_j$ e sua união A. A valoração A satisfaz $f_c(G, V)$ como mostra o teorema 5.

Como toda aresta de G' está presente em algum P'_i, e como a valoração A associa o valor verdade 1 às variáveis correspondentes às arestas que pertencem aos caminhos P'_i, todas as arestas de G' estão presentes em $G_H(A)$. Além disso, como todo caminho P'_i contém apenas arestas de G', e como $G_H(A)$ contém exatamente as arestas de todos os

caminhos P'_i, tal grafo não contém nenhuma aresta que não pertence a G'. Dessa forma, $G_H(A)$ contém exatamente as arestas de G', e, logo, $G_H(A) = G'$.

Todo vértice v_{c_i} de G' está diretamente conectado a exatos dois outros vértices no mesmo grafo, $v_{c_{i-1}}$ e $v_{c_{i+1}}$, através das arestas $e_{i-1} = \{v_{c_{i-1}}, v_{c_i}\}, e_i = \{v_{c_i}, v_{c_{i+1}}\} \in E(G')$. Como e_{i-1} e e_i estão em $G_H(A)$, $f_{e_{i-1}}$ e f_{e_i} são satisfeitas por A, e essas arestas são as únicas da vizinhança de v_{c_i} em $G_H(A)$.

Assim, $C_2(\{f_{e_j} : e_j \in N(v_i)\})$ é satisfeito para todo v_i, e logo a fórmula $f_H(G)$ é satisfeita por A. □

Teorema 10: Seja G um grafo conexo com $|V| \geq 2$ vértices. Se há 2 vértices $v_a, v_b \in V$ com grau 1 e $|V| - 2$ vértices com grau 2, então G consiste exatamente em um caminho Hamiltoniano (que contém todos os vértices de G) entre v_a e v_b.

Prova: Semelhantemente ao mostrado no teorema 2, existe um caminho P entre v_a e v_b em G, pois ambos os vértices têm grau igual a 1 e todos os demais têm grau igual a 0 ou 2 (grau 2, em particular).

Como G é conexo, todos os vértices de G estão na mesma (e única) componente conexa de P. Assim, para cada vértice $v_i \in V$, existe um caminho P'_i, possivelmente vazio e inteiramente fora de P, entre v_i e um vértice v_j que pertence a P.

Entretanto, se P'_i não é vazio, existiria uma aresta e'_i em P'_i que contém v_j e que não está presente em P. Se $v_j \in \{v_a, v_b\}$, v_j estaria conectado a uma aresta de P e à aresta e'_i. Assim, seu grau seria 2, o que é impossível, dado que o grau de v_a e v_b é 1. Da mesma forma, se $v_j \notin \{v_a, v_b\}$, v_j estaria conectado a duas arestas de P e à aresta e'_i, o que tornaria seu grau igual a 3. Isso também é impossível, pois todos os vértices diferentes de v_a e v_b têm grau 2. Dessa forma, P'_i necessariamente é um caminho vazio, e logo $v_i = v_j$.

Como existe um v_j que está no caminho entre v_a e v_b, e como $v_i = v_j$ para todo $v_i \in V$, todos os vértices de G estão presentes no caminho P. Assim, P é um caminho Hamiltoniano entre v_a e v_b. □

Teorema 11: Se existe um modelo $A : \mathscr{X} \to \mathbb{B}$ para $f_H(G)$, então G é Hamiltoniano e $G_H(A)$ consiste em um ciclo Hamiltoniano.

Prova: Seja $A : \mathscr{X} \to \mathbb{B}$ uma valoração que satisfaz $f_H(G)$. Devido à primeira

restrição da fórmula $f_H(G)$, existe um caminho em $G_H(A)$ entre cada par de vértices adjacentes em uma permutação qualquer de V. Assim, por transitividade, $G_H(A)$ é conexo. Da mesma forma, devido às demais restrições da fórmula, há exatamente duas arestas que contém v_i no subgrafo, para todo $v_i \in V$. Assim, todos os vértices de $G_H(A)$ têm grau igual a 2.

Há $(\sum_{v \in V(G_H(A))} d(v))/2 = (|V| \times 2)/2 = |V| > |V| - 1$ arestas em $G_H(A)$. Além disso, esse grafo é conexo. Logo, devido a propriedades conhecidas de árvores [14], $G_H(A)$ é cíclico, isto é, existe ao menos um ciclo C em $G_H(A)$.

Seja $e_i = \{v_a, v_b\}$ uma aresta qualquer do ciclo C que contém os vértices v_a e v_b. Considere o grafo $G'_H(A, e_i) = (V(G_H(A)), E(G_H(A)) \backslash \{e_i\})$, que consiste no grafo $G_H(A)$ sem a aresta e_i. Como o grau de v_a (v_b) em $G_H(A)$ é igual a 2, e como uma de suas arestas deixa de existir em $G'_H(A, e_i)$, o grau de v_a (v_b) nesse grafo é igual a 1.

Além disso, C é um subgrafo biconexo de $G_H(A)$, isto é, existem dois caminhos entre qualquer par de vértices em C. Dessa forma, a remoção da aresta e_i de C não torna $G_H(A)$ desconexo, pois há outro caminho entre v_a e v_b no grafo, além do caminho que consiste apenas nessa aresta.

Assim, $G'_H(A, e_i)$ é um grafo conexo cujos dois vértices (v_a e v_b) têm grau 1, e os restantes têm grau 2. Pelo teorema 10, esse grafo consiste exatamente em um caminho Hamiltoniano entre v_a e v_b.

Como o grafo $G_H(A)$ consiste na inserção da aresta $e_i = \{v_a, v_b\}$ ao grafo $G'_H(A, e_i)$, que consiste em um caminho Hamiltoniano entre v_a e v_b, então $G_H(A)$ consiste em um ciclo Hamiltoniano (e, logo, G é um grafo Hamiltoniano). □

A fórmula $f_c(C, V)$ tem um número quadrático de elementos no tamanho do grafo dado, como apresentado na seção 3.2.3. Como a aridade do operador de disjunção utilizado pela fórmula f_{e_j} é a igual a $|V|-1$, há $O(|V|)$ elementos na fórmula f_{e_j}, para todo $e_j \in E$. Assim, o tamanho da subfórmula $C_2(\{f_{e_j} : e_j \in N(v_i)\})$ é proporcional a $|V|$ vezes o grau de v_i no grafo G. Dessa forma, o número de elementos de tais subfórmulas pode ser dado por $(|V|-1) \times 2|E|$, que também é quadrático em $|V|+|E|$. Assim, o tamanho da fórmula $f_H(G)$ também é quadrático no tamanho do grafo dado.

3.2.6 Ciclo Hamiltoniano Mínimo

O problema do Ciclo Hamiltoniano Mínimo pode ser reduzido para MaxSAT se a fórmula $f_H(G)$, definida pela equação 3.6, for classificada como *hard*. As fórmulas *soft* podem ser dadas por $f_{s_i} = \neg f_{e_i}$, cujo peso é $w(e_i)$, para cada $e_i \in E(G)$. A fórmula f_{e_i} é definida pela equação 3.5. A instância de MaxSAT é então dada por:

$$\begin{aligned} f_h &: f_H(G) \\ f_s &: \{(\neg f_{e_i}, w(e_i)) : e_i \in E(G)\} \end{aligned} \tag{3.7}$$

A corretude da redução é dada pelos seguintes teoremas:

Teorema 12: Seja G um grafo Hamiltoniano. Se $A : \mathscr{X} \to \mathbb{B}$ é um modelo ótimo para a instância gerada, então $G_H(A)$ consiste em um ciclo Hamiltoniano mínimo de G.

Prova: Seja $A : \mathscr{X} \to \mathbb{B}$ um modelo ótimo para a instância gerada para o grafo G. O subgrafo $G_H(A)$ consiste em um ciclo Hamiltoniano de G, como mostrado pelo teorema 11.

Além disso, a fórmula *soft* f_{s_i} é verdadeira se e somente se a aresta $e_i \in E(G)$ não pertence a $G_H(A)$. Dessa forma, A satisfaz as (e apenas as) fórmulas *soft* que correspondem as arestas ausentes de $G_H(A)$.

Por A ser um modelo ótimo, a soma dos pesos das fórmulas *soft* satisfeitas por A é máxima, e, logo, a soma dos pesos das arestas presentes de $G_H(A)$ é mínima. Assim, $G_H(A)$ consiste em um ciclo Hamiltoniano mínimo de G. □

Teorema 13: Se G' é um ciclo Hamiltoniano mínimo de um grafo ponderado G, então existe um modelo $A : \mathscr{X} \to \mathbb{B}$ que é solução da instância gerada de G.

Prova: Seja G um grafo ponderado Hamiltoniano, e G' um ciclo Hamiltoniano mínimo de G.

Considere a valoração $A : \mathscr{X} \to \mathbb{B}$ descrita no teorema 9, que associa valores verdades às variáveis de tal forma que $G_H(A) = G'$.

Como G' é um ciclo Hamiltoniano mínimo, a soma dos pesos de suas arestas é mínima.

Além disso, como f_{s_i} é satisfeita por A se e somente se a aresta $e_i \in E(G)$ está ausente de $G_H(A)$, a soma dos pesos das fórmulas *soft* satisfeitas por A é máxima.

Logo, o modelo A é uma solução de MaxSAT para a instância gerada. □

A fórmula *hard* gerada tem tamanho quadrático no tamanho do grafo original, como mostrado na seção 3.2.5. Além disso, cada uma das $|E|$ fórmulas *soft* em f_s tem tamanho linear em $|V|$, como também apresentado. Logo, a fórmula gerada por esta redução também tem tamanho quadrático no tamanho do grafo dado.

3.2.7 Clique

Clique pode ser reduzido a SAT através da construção da fórmula $f_K(G, k)$, que é satisfatível se e somente se o grafo G contém uma clique de tamanho k. Além disso, todo modelo A para $f_K(G, k)$ descreve um subgrafo $G_K(A)$ de G que consiste na clique buscada. Esta fórmula pode ser construída da seguinte maneira:

- De forma análoga à redução de Caminho Mínimo, associa-se uma variável binária x_i para cada aresta $e_i \in E(G)$. Seja $\mathscr{X} = \{x_i : e_i \in E(G)\}$ o conjunto dessas variáveis. Dado um modelo $A : \mathscr{X} \to \mathbb{B}$ para $f_K(G, k)$, $G_K(A)$ é definido como o subgrafo de G induzido pelo conjunto de arestas $\{e_i \in E(G) : A(x_i) = 1\}$;
- Deve haver exatamente k vértices em $G_K(A)$ cujo grau é $k-1$. Esta necessidade pode ser codificada com a restrição $C_k(C_{k-1}(\mathscr{N}(v_1)), C_{k-1}(\mathscr{N}(v_2)), ...C_{k-1}(\mathscr{N}(v_{|V|})))$, onde $\{v_1, ..., v_{|V|}\} = V(G)$;
- Além disso, os outros $|V| - k$ vértices devem ter grau 0 em $G_K(A)$. Isto pode ser feito com a restrição $C_{|V|-k}(C_0(\mathscr{N}(v_1)), C_0(\mathscr{N}(v_2)), ..., C_0(\mathscr{N}(v_{|V|})))$.

Dessa forma, a fórmula $f_K(G, k)$ pode ser construída como a conjunção das duas restrições citadas:

$$f_K(G,k) = C_k(C_{k-1}(\mathscr{N}(v_1)), C_{k-1}(\mathscr{N}(v_2)), ..., C_{k-1}(\mathscr{N}(v_{|V|}))) \wedge C_{|V|-k}(C_0(\mathscr{N}(v_1)), C_0(\mathscr{N}(v_2)), ..., C_0(\mathscr{N}(v_{|V|}))) \tag{3.8}$$

A corretude desta redução é mostrada pelos seguintes teoremas:

Teorema 14: Se existe uma clique de tamanho k em G, então $f_K(G,k)$ é satisfatível.

Prova: Seja G' uma clique de tamanho k em G, e seja A a valoração que associa 1 às variáveis correspondentes às arestas de G', isto é, $A(x_i) = 1$ se e somente se $e_i \in E(G')$.

Como G' é um subgrafo completo, todos os vértices de $V(G')$ tem o mesmo grau, que é o maior possível, igual a $|V(G')| - 1$. Como G' tem $|V(G')| = k$ vértices, o grau de todos os seus vértices é igual $k-1$. Logo, existem exatamente $k-1$ arestas na vizinhança do vértice v_i na clique, para todo $v_i \in V(G')$. Como isso ocorre com exatos k vértices, A satisfaz a primeira restrição de $f_K(G,k)$.

Além disso, todos os outros $|V|-k$ vértices em G não pertencem a G', e logo nenhuma (0) aresta de suas vizinhanças estão na clique. Assim, A também satisfaz a segunda restrição de $f_K(G,k)$, e, logo, A satisfaz toda a fórmula. □

Teorema 15: Se $f_K(G,k)$ é satisfatível, então G contém uma clique de tamanho k.

Prova: Seja A um modelo para $f_K(G,k)$. Como A satisfaz a segunda restrição da fórmula, existem pelo menos $|V| - k$ vértices para os quais nenhuma aresta de $G_K(A)$ os contém, por seus graus serem iguais a zero. Assim, existem no máximo k vértices em $G_K(A)$.

Além disso, como A satisfaz a primeira restrição da fórmula, existem no mínimo k vértices para os quais existe uma ou mais arestas em $G_K(A)$ que os contém, já que seus graus são iguais a $k-1$ e assume-se que $k > 1$. Considerando ambas as observações, conclui-se que existem exatamente k vértices em $G_K(A)$.

Pela primeira restrição da fórmula, o grau de cada vértice em $G_K(A)$ é igual a $k-1$. Este grau é máximo, dado que existem k vértices em tal grafo. Dessa forma, cada vértice de $G_K(A)$ está conectado a todos os outros $k-1$ vértices do subgrafo.

Assim, $G_K(A)$, subgrafo de G, é uma clique de tamanho k. □

O número de elementos na primeira e na segunda restrição de $f_K(G,k)$ é proporcional ao número de vértices em G, mais a soma dos graus de todos os seus vértices, $2|E|$. Precisamente, existem $4|E|+2|V|+3$ elementos em toda a fórmula, que usa $|E|$ variáveis. Logo, o tamanho da fórmula é linear no tamanho do grafo dado.

3.3 Considerações

Esse capítulo apresenta reduções de problemas em grafos para SAT ou MaxSAT, tanto algumas conhecidas quanto as novas descritas neste trabalho. É válido citar que parte das novas reduções foi publicada recentemente pelos autores desse trabalho em um congresso da área [16].

Todas as novas reduções apresentadas são lineares ou quadráticas no tamanho do grafo dado, o que as torna assintoticamente semelhantes ou menores do que as reduções publicadas anteriormente.

Essas reduções formam uma das duas principais contribuições deste trabalho. A outra consiste na modificação de um resolvedor SAT para tratar as instâncias geradas por elas. O capítulo 4 apresenta o funcionamento de resolvedores SAT e MaxSAT eficientes, enquanto o capítulo 5 apresenta tal modificação.

CAPÍTULO 4

MÉTODOS DE RESOLUÇÃO DE SATISFABILIDADE BOOLEANA E MÁXIMA

A segunda principal contribuição deste trabalho é a adaptação de um resolvedor SAT e MaxSAT para resolver os problemas reduzidos no capítulo 3. Assim, é necessário apresentar os métodos conhecidos para a resolução de tais problemas e as modificações realizadas em dados resolvedores.

Este capítulo apresenta uma revisão bibliográfica dos algoritmos conhecidos para a resolução dos problemas SAT e MaxSAT, utilizados pelos resolvedores no estado-da-arte. Suas duas subseções apresentam os algoritmos utilizados para SAT e MaxSAT, respectivamente.

4.1 O problema de Satisfabilidade Booleana

Esta seção apresenta técnicas utilizadas para se resolver o problema SAT. Os algoritmos completos mais eficientes conhecidos baseiam-se no procedimento DPLL, apresentado a seguir.

4.1.1 O procedimento DPLL

O algoritmo mais conhecido para a resolução do problema SAT é o DPLL [15], descrito pelo algoritmo 1.

É mantida durante a execução do algoritmo uma valoração A, que conterá o modelo procurado caso a fórmula de entrada seja satisfatível. Inicialmente, $A = \emptyset$. O procedimento, recursivo baseado em retrocesso, consiste em escolher uma variável que aparece em f (linha 5) e associá-la ao valor verdade 1, formando-se um literal que representa a associação realizada (x_i). O literal é então adicionado à A, a valoração parcial atual (linha

Entrada: Uma fórmula f
Saída: *Satisfatível* ou *Não Satisfatível*
1 **se** $f = 0$ **então**
2 **retorna** Não satisfatível
3 **se** $f = 1$ **então**
4 **retorna** Satisfatível
5 Escolhe uma variável x_i que aparece em f
6 $A = A \cup \{x_i\}$
7 **se** $DPLL(BCP(f, x_i))$ =*Satisfatível* **então**
8 **retorna** Satisfatível
9 $A = (A \setminus \{x_i\}) \cup \{\neg x_i\}$
10 **se** $DPLL(BCP(f, \neg x_i))$ =*Satisfatível* **então**
11 **retorna** Satisfatível
12 **retorna** *Não satisfatível*

Algoritmo 1: DPLL

6).

Ocorre então a *propagação* (*Boolean Constraint Propagation* ou BCP()) do literal na fórmula (linha 7). Durante a propagação, as instâncias da variável escolhida são trocadas pelo seu valor verdade correspondente, e operadores cujo valor verdade passa a ser conhecido também são substituídos. O problema é então resolvido recursivamente para a fórmula resultante (linha 7). Se a fórmula é satisfatível, f também o é (linha 8). Caso contrário, o valor verdade 0 é associado à variável x_i e o processo se repete de forma análoga (linhas 9 a 11).

Se as fórmulas resultantes das propagações não são satisfatíveis, então certamente a fórmula de entrada também não é, uma vez que todos os possíveis valores verdades para uma dada variável se esgotaram. Assim, o algoritmo retorna *Não satisfatível* (linha 12). O algoritmo potencialmente testa todas as $2^{|\mathscr{X}|}$ possíveis valorações utilizando a técnica de retrocesso, sendo $\mathscr{X}$ o conjunto de variáveis da fórmula de entrada.

Quando a fórmula de entrada está em CNF, duas regras podem ser utilizadas para melhorar o desempenho do algoritmo. A regra do *literal puro* consiste em verificar se uma dada variável x_i aparece na fórmula com apenas um de seus literais, isto é, verificar se x_i aparece e $\neg x_i$ não aparece ou vice-versa. Neste caso, é possível forçar (ou *implicar*) o valor verdade da variável de tal forma que seu literal se torne verdadeiro. A regra da *cláusula*

unitária consiste em verificar se, em qualquer ponto do algoritmo, alguma cláusula C_j é composta por somente um literal x_i, isto é, $C_j = \{x_i\}$, para algum $C_j \in f$. Neste caso, também é possível implicar $sig(x_i)$ como valor verdade de $var(x_i)$. Ambas as regras são aplicadas durante a propagação de um literal.

O algoritmo 2 descreve o processo de propagação em uma fórmula em CNF. Se uma cláusula C contém o literal x_i propagado, seu valor verdade torna-se conhecido e igual a 1 e, como $(f \setminus \{C\}) \wedge 1 = (f \setminus \{C\})$, a cláusula pode ser removida da fórmula (linhas 1 e 2). Se uma cláusula contém a negação do literal em questão, o literal naquela cláusula assume valor 0 e, como $(C \setminus \{x_i\}) \vee 0 = (C \setminus \{x_i\})$, este pode ser removido da cláusula (linhas 3 e 4). As regras de literal puro e cláusula unitária são então aplicadas (linhas 5 e 6). Se f contém uma cláusula vazia (denotada por $\square$) indicando que todos os seus literais foram removidos, então o valor verdade da cláusula se torna 0, o que torna a fórmula insatisfatível (linhas 7 e 8). A ausência de cláusulas na fórmula indica que todas as cláusulas foram removidas e, portanto, satisfeitas, o que indica que a fórmula é satisfatível (linhas 9 e 10).

```
   Entrada: Uma fórmula f e um literal x_i
   Saída: Uma fórmula f' resultante da propagação de x_i
 1 para cada C ∈ f : x_i ∈ C faça
 2 |  f = f \ {C}
 3 para cada C ∈ f : ¬x_i ∈ C faça
 4 |  C = C \ {¬x_i}
 5 se Existe literal puro x_j ou cláusula unitária {x_j} em f então
 6 |  retorna BCP(f, x_j)
 7 se f contém cláusula vazia então
 8 |  retorna 0
 9 se f não contém cláusulas então
10 |  retorna 1
11 retorna f
```

Algoritmo 2: BCP() em fórmulas CNF

4.1.2 Técnicas para melhoria do desempenho do DPLL

O algoritmo DPLL, embora seja completo e correto, é ineficiente. Assim, melhorias no procedimento foram propostas posteriormente, principalmente para fórmulas CNF.

O resolvedor SATO [49] propõe uma estrutura de dados para representar cláusulas que melhora o desempenho da função $BCP()$. Cada cláusula é representada por uma lista de literais, e são mantidos para cada cláusula dois ponteiros para a lista, inicialmente para o primeiro e o último elemento da mesma. Os literais para os quais ponteiros apontam são ditos *literais vigiados*.

Quando o literal propagado é a negação de um literal vigiado, o ponteiro simplesmente avança uma posição na lista (se for o primeiro ponteiro da cláusula, o ponteiro passa a apontar para a próxima posição; caso contrário, passa a apontar para a posição anterior). Quando ambos os ponteiros apontam para o mesmo literal, a cláusula se torna unitária, possibilitando a inferência do valor verdade do literal. Embora simples, esta estrutura dá ao DPLL um grande aumento de desempenho, uma vez que o procedimento $BCP()$ se torna computacionalmente barato em cada cláusula, e a fórmula não precisa ser reconstruída a cada passo do algoritmo.

Também para acelerar o processo de busca de um resolvedor SAT, foram propostas as ideias de *aprendizado* e *retrocesso não cronológico*, primeiramente introduzidas no resolvedor GRASP [36].

Cada variável pode ter um valor verdade associado de duas maneiras: Através da escolha de seu valor pelo processo DPLL (uma *decisão*) ou através da inferência de seu valor devido a uma valoração anterior (uma *implicação*). Nos algoritmos apresentados, variáveis decididas são as escolhidas na linha 5 do algoritmo 1, enquanto variáveis implicadas são as obtidas pelo uso das regras na linha 6 do algoritmo 2.

Pode-se associar, para cada variável x_i decidida, o *nível de decisão* $\delta(x_i)$ da variável, que é o nível no processo de busca em que o valor da variável foi escolhido. Dessa forma, a primeira variável decidida tem nível 0, a segunda 1, a próxima 2, e assim por diante. O processo de retrocesso do DPLL consiste em retornar de um nível de decisão para outro imediatamente anterior. Assim, se o processo sofrer retrocesso durante a propagação da variável decidida no nível 1, seu valor é trocado e a próxima variável decidida também tem nível de decisão 2.

Seja $A(x_i)$ o conjunto de literais ou, equivalentemente, uma valoração parcial que

implica em um valor para x_i. Se a fórmula possui, por exemplo, a cláusula $(x_1 \vee x_2 \vee \neg x_3)$, tem-se que $A(x_1) = \{(x_2, 0), (x_3, 1)\}$, $A(x_2) = \{(x_1, 0), (x_3, 1)\}$ e $A(x_3) = \{(x_1, 0), (x_2, 0)\}$. $A(x_1)$ pode também ser escrito como $A(x_1) = \{\neg x_2, x_3\}$.

O nível de decisão de uma variável x_i implicada é então definido como o maior nível de decisão das variáveis em $A(x_i)$, isto é, $\delta(x_i) = max\{\delta(x_j) : x_j \in A(x_i)\}$.

Durante o processo de propagação de valores de variáveis implicadas, pode-se construir um *digrafo de implicações* da seguinte maneira:

- Cada vértice corresponde à associação de uma variável x_i a um valor verdade $v(x_i)$ no nível de decisão $\delta(x_i)$, representado por $x = v(x)@\delta(x)$;

- Os antecessores de um vértice x_i são as variáveis da valoração parcial $A(x_i)$ que levou x_i a ser implicado, por ter se tornado a única variável de uma cláusula unitária C. Os arcos do digrafo são rotuladas com C. Vértices sem antecessores representam variáveis decididas;

- Um vértice κ corresponde a um *conflito*, isto é, uma cláusula C que se tornou falsa devido à valoração parcial gerada. $A^0(\kappa)$ é definido como a valoração que tornou aquela cláusula falsa.

A figura 4.1 exemplifica a construção do digrafo, considerando a fórmula composta pelas cláusulas $C_1 = \neg x_1 \vee x_2$, $C_2 = \neg x_1 \vee x_3 \vee x_9$, $C_3 = \neg x_2 \vee \neg x_3 \vee x_4$, $C_4 = \neg x_4 \vee x_5 \vee x_{10}$, $C_5 = \neg x_4 \vee x_6 \vee x_{11}$, $C_6 = \neg x_5 \vee \neg x_6$, $C_7 = x_1 \vee x_7 \vee \neg x_{12}$, $C_8 = x_1 \vee x_8$, $C_9 = \neg x_7 \vee \neg x_8 \vee \neg x_{13}$, e considerando a valoração parcial atual $\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, x_1 = 1@6\}$.

Nota-se que, com o digrafo, é possível obter uma valoração parcial $A^n(\kappa)$ que implica, direta ou indiretamente, no conflito κ. Como tal valoração impede que a fórmula seja satisfeita, sua negação deve obrigatoriamente ser satisfeita pela valoração buscada. Assim, uma cláusula $C(\kappa)$ contendo a negação de $A^n(\kappa)$, é *aprendida*, isto é, é adicionada na fórmula. Para se obter $A^n(\kappa)$, particiona-se $A(x_i)$, para todo x_i, em:

- $\Lambda(x_i) = \{(y, v(y)) \in A(x_i) : \delta(y) < \delta(x_i)\}$, as variáveis de $A(x_i)$ cujo nível de decisão é estritamente menor que o de x_i;

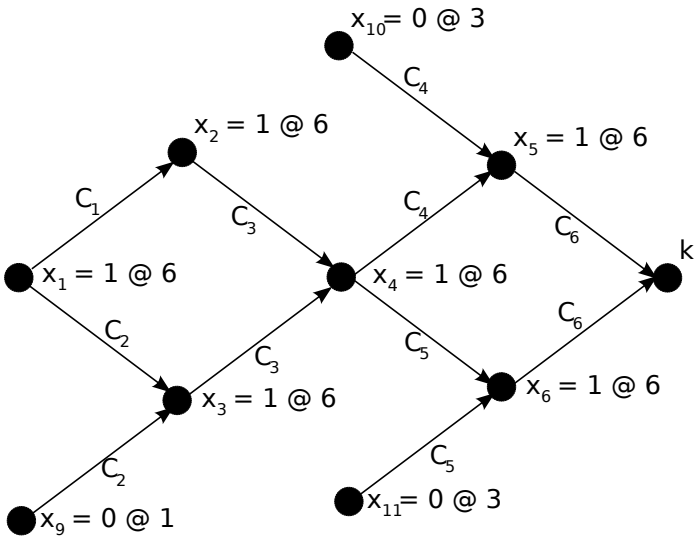

Figura 4.1: Exemplo da construção do digrafo de implicações

- $\Sigma(x_i) = \{(y, v(y)) \in A(x_i) : \delta(y) = \delta(x_i)\}$, as variáveis de $A(x_i)$ cujo nível de decisão é igual ao de x_i.

Uma possível valoração $A^n(\kappa)$ pode então ser calculada da seguinte maneira:

- $A^n(x_i) = (x_i, v(x_i))$ se x_i foi decidida;
- $A^n(x_i) = \Lambda(x_i) \cup [\cup_{(y,v(y))\in\Sigma(x_i)} A^n(y)]$ se x_i foi implicada ou se $x_i = \kappa$.

Tal cálculo corresponde a uma busca para trás no digrafo de implicações, iniciando-se em κ. Em cada vértice correspondente a uma valoração $x_i = v(x_i)$, são adicionadas em $A^n(\kappa)$ as variáveis de $A(x_i)$ valoradas em níveis de decisões anteriores ao nível de x_i, e a busca é recursivamente usada para os vértices cujo níveis de decisão são iguais ao do vértice atual. No exemplo, é aprendida a cláusula $C(\kappa) = \neg(\neg x_{10} \wedge \neg x_{11} \wedge \neg x_9 \wedge x_1) = (x_{10} \vee x_{11} \vee x_9 \vee \neg x_1)$.

Nota-se que, se todas as variáveis de $A^n(\kappa)$ têm níveis de decisões estritamente menores que o último nível, então $A^n(\kappa)$ também implica em um conflito κ' em um nível de decisão menor, já que as variáveis valoradas entre este nível e o último não são de fato relevantes para implicar o conflito. Assim, o digrafo de implicações pode ser alterado, fazendo com que o conflito κ' seja representado anteriormente a κ.

Além disso, o procedimento de busca não irá encontrar uma valoração para a fórmula

enquanto não voltar, por retrocesso, a um nível de decisão β que tenha influência em $A^n(\kappa')$. O valor de β pode ser calculado como $\beta = max\{\delta(x_i) : (x_i, v(x_i)) \in A^n(\kappa')\}$, o maior nível de decisão onde alguma variável de $A^n(\kappa')$ foi valorada. Assim, o processo de busca pode seguramente realizar o retrocesso até o nível β de decisão.

Se β for ainda menor que o penúltimo nível de decisão, tal retrocesso é dito *retrocesso não cronológico*, já que potencialmente volta vários níveis de decisão atrás, e não apenas um, como nas máquinas de busca tradicionais.

Embora as ideias de aprendizado e de retrocesso não cronológico sejam interessantes, alguns problemas podem surgir. Primeiramente, o custo para se manter as estruturas auxiliares durante o processo de busca pode ser inviável, para algumas instâncias de SAT. Além disso, o número de cláusulas aprendidas pode, no pior caso, ser exponencial.

O último problema pode ser amenizado limitando-se o número e o tamanho das cláusulas aprendidas. Dado um $k \in \mathbb{N}$ fixo, pode-se aprender cláusulas apenas de tamanho menor ou igual a k. Além disso, pode haver mais de uma valoração $A_C(\kappa)$ possível para um dado κ.

Uma valoração pode ser obtida através do cálculo dos dominantes do vértice κ no digrafo de implicações. Tais dominantes, chamados *Unique Implication Points* (UIPs), podem ser encontrados em tempo linear no digrafo de implicações [36]. Por fim, se uma cláusula C_i *subjuga* uma cláusula C_j (isto é, se $C_j \subseteq C_i$), C_i pode ser removida da fórmula, uma vez que se C_j é satisfeita, então C_i certamente também o é. A operação de subjugação é computacionalmente cara e não é tratada nos resolvedores SAT atuais.

As ideias propostas no GRASP foram de fato inovadoras e importantes para o desenvolvimento dos resolvedores SAT. Atualmente, não há um resolvedor SAT completo no estado-da-arte que não utilize as técnicas propostas pelos autores do GRASP. Embora originalmente trabalhe em instâncias SAT apenas em CNF, as mesmas técnicas podem ser utilizadas para instâncias SAT não clausais, já que o aprendizado e o retrocesso não cronológico derivam apenas das variáveis decididas durante o processo.

Há ainda outros resolvedores que propõem ideias para a melhoria do desempenho do algoritmo DPLL, como os reinícios, que consistem em reinicializar todo o procedimento

do DPLL em dados instantes, mantendo-se porém as cláusulas aprendidas em execuções anteriores [26].

Além disso, a escolha de uma variável decidida pode ser feita por alguma heurística que tenta antecipar as podas no espaço de busca (*heurística de ramificação*). As heurísticas Bohm [12], MOM [20] e Jeroslow-Wang [28], por exemplo, tendem a escolher as variáveis que aparecem em cláusulas de tamanho pequeno. A heurística MOM, por exemplo, escolhe as variáveis que mais ocorrem nas cláusulas de tamanho igual ao tamanho da menor cláusula da fórmula, em um dado passo.

Essas heurísticas de decisão visam fazer com que cláusulas unitárias e, consequentemente, inferências sejam obtidas mais rapidamente. Há também a heurística VSIDS [37], que dinamicamente aproveita o método de aprendizado para escolher variáveis com maior frequência na fórmula. A heurística VSIDS é a mais utilizada em resolvedores SAT genéricos por apresentar um bom desempenho em instâncias SAT que não pertencem a um domínio de aplicação muito específico.

4.1.3 Satisfabilidade Não Clausal

Os resolvedores mais eficientes atuais exigem que a fórmula de entrada esteja em CNF. Dessa forma, quando se necessita resolver uma instância não clausal de SAT, é comum convertê-la para CNF. Entretanto, resolvedores capazes de trabalhar diretamente com instâncias não clausais foram desenvolvidos. Dentre eles, destaca-se o LIAMFSAT, um resolvedor capaz de verificar fórmulas no formato não clausal ISCAS [41].

Uma fórmula em formato ISCAS consiste em um Grafo Dirigido Acíclico (DAG) onde as folhas representam as variáveis e nós internos representam operadores. Por ser unário, o operador de negação em particular não é representado por um nó, mas sim por um rótulo nos arcos da árvore. A fórmula é então representada pela raiz do DAG. A figura 4.2 exemplifica a representação de uma fórmula no formato ISCAS.

O formato de entrada de uma fórmula em ISCAS, semelhante à descrição de um circuíto lógico, é descrito pelas seguintes expressões:

- `INPUT(x)`

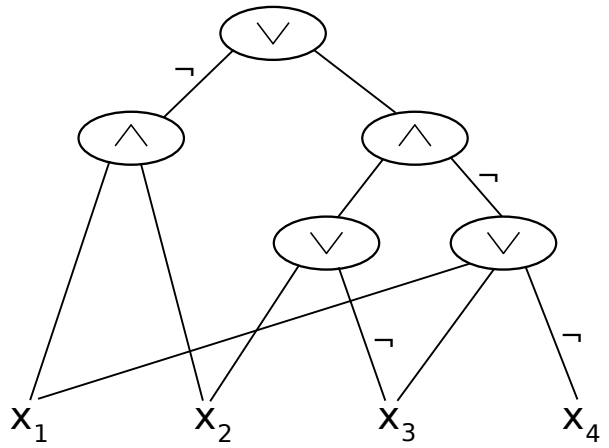

Figura 4.2: Formato ISCAS da fórmula $\neg(x_1 \wedge x_2) \vee ((x_2 \vee \neg x_3) \wedge \neg(x_1 \vee x_3 \vee \neg x_4))$

indica que x é uma variável da fórmula;

- `OUTPUT(g)`

  indica que a saída da porta g representa a raiz da fórmula;

- `g = AND(g1,g2,...,gk)`

  indica que a saída da porta g consiste na conjunção da saída das portas $g1...gk$;

- `g = OR(g1,g2,...,gk)`

  indica que a saída da porta g consiste na disjunção da saída das portas $g1...gk$;

- `g = NOT(g1)`

  indica que a saída da porta g consiste na negação da saída da porta $g1$.

A fórmula apresentada na figura 4.2 pode ser descrita pela seguinte entrada:

```
INPUT(x1)      nx3 = NOT(x3)         ng3 = NOT(g3)
INPUT(x2)      nx4 = NOT(x4)         g4 = AND(g2,ng3)
INPUT(x3)      g1 = AND(x1,x2)       ng1 = NOT(g1)
INPUT(x4)      g2 = OR(x2,nx3)       out = OR(ng1,g4)
OUTPUT(out)    g3 = OR(x1,x3,nx4)
```

O LIAMFSAT, que utiliza o formato ISCAS, tem como procedimento base o algoritmo DPLL. A cada passo, uma variável é decidida e seu valor verdade escolhido é propagado na fórmula, por sua função $BCP()$. A propagação consiste em rotular o nó correspondente à variável no DAG com o seu valor verdade associado, e empilhar seus pais em uma pilha de nós. A cada nó desempilhado, então, é verificado se seu valor verdade já é conhecido, através dos valores verdade de seus filhos. Em caso positivo, seus pais são empilhados, e assim sucessivamente. Se, após a propagação, a raiz do DAG se torna verdadeira, um modelo é encontrado e o algoritmo para. Caso a raiz se torne falsa, um conflito é detectado.

No LIAMFSAT, todas as variáveis são de decisão. Além disso, embora não sejam variáveis, todos os operadores têm seu valor verdade *implicados* por uma valoração parcial. Desta forma, em caso de detecção de conflito, o próprio DAG da fórmula pode ser utilizado como digrafo de implicações e usado para se obter uma cláusula aprendida, como no GRASP. Isto evita então a necessidade da criação e manutenção do grafo auxiliar. Quando uma cláusula é aprendida, ela é anexada à raiz da fórmula através de uma conjunção.

Embora realize as técnicas de aprendizado e retrocesso não cronológico, o LIAMFSAT não realiza regras de inferências. Embora a fórmula de entrada não esteja em CNF, todo o aprendizado obtido pelo LIAMFSAT resulta em cláusulas, que são anexadas à fórmula não clausal. O resolvedor não aproveita o fato de que a parte aprendida da fórmula está em CNF para melhorar seu desempenho. Além disso, o resolvedor não faz reinícios, e não aplica heurísticas de ramificação.

Além do LIAMFSAT, outros resolvedores SAT não clausal podem ser citados, como o NOCLAUSE [44]. O NOCLAUSE também atua com fórmulas no formato ISCAS e também usa o DPLL como procedimento base. Entretanto, diferentemente do LIAMFSAT, os operadores podem ter seus valores verdade decididos, assim como as variáveis. A cada passo do algoritmo, um vértice do DAG da fórmula (seja ele uma variável ou operador) é escolhido e seu valor verdade é decidido e propagado. O retrocesso deve ocorrer quando a raiz do DAG recebe o valor verdade 0, ou quando a função $BCP()$ decide propagar a um nó o valor verdade oposto ao valor já decidido para aquele nó.

Enquanto o procedimento de propagação do LIAMFSAT percorre e empilha apenas os pais dos nós cujos valores verdade são conhecidos, o NOCLAUSE pode percorrer tanto os pais quanto os filhos de um nó. Tais propagações ocorrem quando o valor de um nó pode ser conhecido através da análise dos valores dos nós próximos, antecessores ou sucessores. Por exemplo, considere um operador de disjunção cujos filhos são duas subfórmulas f_1 e f_2, onde f_1 já foi valorado como falso. Se o procedimento de busca escolhe este operador e decide valorá-lo como verdadeiro, f_2 necessariamente deve ser verdadeiro. Logo, o valor verdade 1 pode ser propagado para f_2. O NOCLAUSE implementa estruturas semelhantes à utilizada na técnica de literais vigiados para melhorar o desempenho da sua função $BCP()$.

Há também resolvedores SAT não clausal que não atuam diretamente no formato ISCAS, mas sim em outras formas de representação da fórmula, como a Forma Normal Negada (*Negation Normal Form* ou NNF). Uma fórmula da lógica proposicional está em NNF se cada operador de negação conecta uma variável, isto é, se a subfórmula de todas as subfórmulas que usam o operador $\neg$ é uma variável. Dentre os resolvedores que trabalham com tal formato destaca-se o NFLSAT [27].

O procedimento principal do NFLSAT consiste na tradução da fórmula NNF em um grafo direcionado cujos vértices representam ocorrências de literais na fórmula e cujos arcos são construídos de acordo com a estrutura da fórmula NNF e dos operadores que conectam seus literais [27]. Tal grafo é construído de forma que a (in)existência de um caminho entre vértices sem arcos de entrada para vértices sem arcos de saída prova a (in)satisfabilidade da fórmula original [27]. Tal caminho é então buscado pelo resolvedor.

4.2 O problema de Satisfabilidade Máxima

Esta seção apresenta técnicas utilizadas para se resolver o problema MaxSAT clausal, o PWMaxSAT. Embora as modificações apresentadas no capítulo 5 baseiam-se em resolvedores não clausais, as técnicas utilizadas são semelhantes às apresentadas nessa seção.

4.2.1 O procedimento DPLL como ramificação e poda

O método mais intuitivo para se resolver o problema MaxSAT consiste em modificar o algoritmo DPLL para fazê-lo agir como um método de ramificação e poda (*Branch and Bound*). Os resolvedores MaxSAT Lazy [1] e MiniMaxSAT [24] utilizam esta técnica. Maximizar a soma dos pesos das cláusulas satisfeitas é matematicamente equivalente a minimizar a soma dos pesos das cláusulas não satisfeitas. Assim, os resolvedores buscam a valoração que minimiza tal soma.

Uma variável global denominada *Cota Superior* (CS) é mantida durante todo o processo. CS armazena, ao final do algoritmo, a soma procurada. A cada fórmula f a ser gerada durante o DPLL é associada uma *Cota Inferior* $CI(f)$ que equivale à soma dos pesos das cláusulas certamente não satisfeitas (vazias) de f.

```
   Entrada: Uma fórmula f
 1 se f_h = 0 então
 2 |  retorna
 3 Ci = CI(f)
 4 se A é valoração completa então
 5 |  CS = min{CS, Ci}
 6 |  retorna
 7 se Ci ≥ CS então
 8 |  retorna
 9 Escolhe uma variável x_i que aparece em f
10 A = A ∪ {x_i}
11 DPLLRP(BCP(f, x_i))
12 A = (A \ {x_i}) ∪ {¬x_i}
13 DPLLRP(BCP(f, ¬x_i))
```

Algoritmo 3: DPLL com Ramificação e Poda (DPLLRP) para MaxSAT

O algoritmo 3 descreve o procedimento. Primeiramente, CS é inicializado com ∞. Então, a cada passo do algoritmo, a cota inferior da fórmula é calculada (linha 3). Se a valoração atual é completa, a cota superior é atualizada em relação à cota inferior calculada (linhas 4 a 6). Além disso, se a cota inferior não for estritamente menor que a cota superior atual, uma poda no espaço de busca pode ser realizada (linhas 7 e 8). Isto se deve ao fato de que não é possível fazer com que a soma dos pesos das cláusulas não satisfeitas de f seja reduzida a partir da propagação de literais em f, uma vez que tal

soma é composta por cláusulas já consolidadas falsas. Uma variável é então escolhida da mesma maneira que o DPLL clássico (linha 9), ambos seus literais são propagados e as fórmulas geradas são analisadas recursivamente (linhas 11 a 13).

Diferentemente do DPLL clássico, uma valoração parcial que torna a fórmula insatisfatível não retrocede o algoritmo, mas apenas incrementa a cota inferior da fórmula em questão. Nota-se também que a regra da cláusula unitária não pode ser aplicada nesse procedimento. Isto se deve ao fato de que uma cláusula de peso maior do que o peso da cláusula unitária pode ser satisfeita se a última não o for.

4.2.2 Fatores de desempenho do algoritmo de ramificação e poda

Embora o algoritmo 3 apresentado na seção anterior seja correto, é possível otimizá-lo através de modificações em três etapas do processo: o cálculo da cota superior inicial, a escolha de variáveis de decisão, e a estimativa da cota inferior de uma dada fórmula.

Além de ∞, o valor da variável CS pode ser inicializado com o valor da cota inferior de qualquer modelo de f_h. Assim, em um pré-processamento, pode-se tentar encontrar algum modelo de f_h (não necessariamente o ótimo para o problema). No resolvedor Lazy, isto é feito através do algoritmo GSAT [42], descrito pelo algoritmo 4.

```
   Entrada: Uma fórmula f
   Saída: Um valor inicial para CS
1  CS = ∞
2  para k de 1 a NUM_ITS faça
3  |  A = valoração completa aleatória
4  |  para l de 1 a NUM_FLIPS faça
5  |  |  Escolhe aleatoriamente um literal x_i ∈ A
6  |  |  A = (A \ {x_i}) ∪ {¬x_i}
7  |  |  se A satisfaz f_h então
8  |  |  |  f' = Propagação de todos os literais em A na fórmula f
9  |  |  |  CS = min(CS, CI(f'))
10 retorna CS
```

Algoritmo 4: Algoritmo GSAT utilizado para cálculo da cota superior inicial

Inicialmente, CS recebe o valor infinito (linha 1). A cada passo do algoritmo, uma valoração completa A é escolhida arbitrariamente (linha 3). Então, NUM_FLIPS vezes,

um literal de A é escolhido (linha 5) e é substituído por sua negação (realizando uma operação de *flip*) (linha 6). Em seguida, se a valoração A satisfaz todas as cláusulas *hard* da fórmula (linha 7), é gerada a formula f' resultante da propagação de todos os literais de A (linha 8), e CS é reduzido para $CI(f')$ se este for menor (linha 9). Após NUM_FLIPS rodadas, uma nova valoração A é gerada, repetindo-se o processo por NUM_ITS iterações (linha 2). Isto evita que o valor buscado se estabilize em torno do melhor valor obtido a partir de uma única valoração A (um *mínimo local*). Se nenhuma valoração obtida satisfizer f_h, CS é inicializado com ∞, como proposto anteriormente. Os valores de NUM_FLIPS e NUM_ITS são escolhidos de forma empírica pelo resolvedor.

O método GSAT não mantém nenhuma informação referente ao histórico de valorações completas geradas. Isso permite que uma dada valoração possa ser analisada mais de uma vez, o que inutiliza processamento. O método IROTS [43], utilizado pelo resolvedor MiniMaxSAT, tende a amenizar tal problema. O algoritmo é semelhante ao GSAT. Entretanto, quando um literal x_i é escolhido para se realizar a operação de *flip*, é armazenada em uma estrutura auxiliar uma informação que indica que nenhum literal de $var(x_i)$ pode ser escolhido nas TL iterações seguintes. Assim, se um literal x_i é obtido aleatoriamente e $var(x_i)$ não pode ser utilizada na iteração atual, outro literal deve ser escolhido. Isto evita que uma dada valoração seja analisada novamente em um intervalo de no mínimo, TL iterações. Novamente, o valor de TL é escolhido de forma empírica pelo resolvedor.

Durante o processo de ramificação e poda, as variáveis de decisão podem ser heuristicamente escolhidas da mesma forma utilizada em resolvedores SAT. O resolvedor Lazy utiliza a heurística MOM [20], enquanto o resolvedor MiniMaxSAT utiliza uma variação da heurística de Jeroslow-Wang para o problema PWMaxSAT, denominada *Weighted* Jeroslow [23], na qual os pesos das cláusulas influenciam a decisão a ser feita.

Por fim, nota-se que, pelo algoritmo 3, se o incremento na cota inferior atingir a cota superior, um corte no espaço de busca é realizado. Embora a soma dos pesos das cláusulas vazias de f_s seja suficiente para o cálculo da cota inferior de uma fórmula f, algumas características da fórmula, se observadas, podem ser utilizadas para obter-se uma cota inferior melhor.

Seja $ic(f, x_i)$ a soma dos pesos das cláusulas de f_s que se tornam vazias caso o literal x_i seja propagado. A estimativa de Wallace Freuder [46], utilizada pelo resolvedor Lazy, consiste em incrementar a cota inferior em $\sum_{x_i} min\{ic(f, x_i), ic(f, \neg x_i)\}$ unidades. Intuitivamente, a estimativa consiste em calcular, para cada variável x_i, a menor soma de pesos de cláusulas vazias obtida através da propagação do literal x_i e do literal $\neg x_i$.

Além disso, se uma fórmula f possui um par de cláusulas unitárias na forma (x_i, u) e $(\neg x_i, w)$ (o que implicaria em um conflito no problema SAT), a fórmula pode ser *modificada* de forma a possuir as cláusulas $(x_i, u - m)$, $(\neg x_i, w - m)$ e $(\square, m)$, onde $m = min\{u, w\}$. A cláusula vazia implica em um incremento imediato de m unidades na cota inferior da fórmula f. Este método, dito *Resolução de Vizinhança Unitária* (*Unit Neighborhood Resolution* ou UNR) [33], é utilizado pelo resolvedor MiniMaxSAT durante seu processo.

Além do método UNR, o resolvedor MiniMaxSAT também utiliza um método de simulação da regra da cláusula unitária (*Simulation of Unit Propagation*) [24], que consiste em, momentaneamente, classificar *todas* as cláusulas de uma fórmula f como *hard*, o que torna a fórmula uma instância do problema SAT. Então, se possível, a regra da cláusula unitária é aplicada sucessivamente à formula, quantas vezes forem possíveis. Se (e apenas se) um conflito for obtido durante a simulação, a cota inferior pode ser incrementada em m unidades, sendo m o menor peso dentre as cláusulas unitárias obtidas.

Esta estimativa pode ser utilizada porque o conjunto de cláusulas que se tornam unitárias durante o processo não é satisfatível e, logo, certamente se tornarão falsas cláusulas com, no mínimo, m unidades de peso.

4.2.3 Técnicas e problemas relacionados

Além da adaptação do algoritmo DPLL para ramificação e poda, outros procedimentos base para o problema MaxSAT e suas generalizações são propostos.

Um *núcleo insatisfatível* (*UNSAT Core* ou UC) de uma fórmula f em CNF é um subconjunto insatisfatível do conjunto de cláusulas de f. Se insatisfatível, a própria fórmula f é um UC dela mesma, e se f é satisfatível, então f não possui um UC.

Fu & Malik [21] propõem um algoritmo para PMaxSAT (não ponderado) baseado na *remoção* iterativa de UCs de uma fórmula, até que a mesma se torne satisfatível. Um UC é removido da fórmula através do *mascaramento* de suas cláusulas, que consiste em adicionar novas variáveis e inseri-las nas cláusulas de tal forma que o conjunto de cláusulas seja satisfatível [48]. A cada UC removido da fórmula, um contador k é incrementado em uma unidade. Ao fim do algoritmo, o valor de k é provadamente o menor número de cláusulas não satisfeitas da fórmula original [21]. O algoritmo WPM1 [3] é uma variação do algoritmo de Fu & Malik para o problema PWMaxSAT.

Embora o algoritmo WPM1 tenha se mostrado competitivo nas últimas competições de MaxSAT [4], é relevante observar alguns pontos:

1. Tanto no WPM1 quanto no algoritmo de Fu & Malik, o mascaramento realizado para se remover um UC de uma fórmula requer que uma subfórmula não clausal seja anexada à mesma. Os autores de ambas as propostas sugerem que tal subfórmula seja convertida para CNF, ao invés de se utilizar um resolvedor não clausal.

2. Além disso, o processamento necessário em uma iteração do processo é proporcional ao tamanho dos UCs obtidos nas iterações anteriormente, uma vez que o tamanho da fórmula cresce ao longo do tempo. Dessa forma, encontrar um UC pequeno é relevante para a eficiência do algoritmo.

Além dos algoritmos utilizados diretamente na resolução da família de problemas de satisfabilidade booleana, é válido estudar problemas que podem ser polinomialmente reduzidos de ou para MaxSAT. Dessa forma, MaxSAT pode ser resolvido através de reduções e algoritmos utilizados para outros problemas, como o de Cobertura Binária.

Seja f uma fórmula booleana sobre um conjunto $\mathscr{X} = \{x_1, ..., x_n\}$ de variáveis booleanas, não necessariamente clausal. Seja também $W : \mathscr{X} \to \mathbb{N}$ uma função de ponderação que associa um peso *natural* a cada *variável* de $\mathscr{X}$. O Problema de Cobertura Binária (*Binate Covering Problem* ou PCB) consiste em maximizar a função $Z = \sum_{i=1}^{n} W(x_i) \times x_i$ sujeito a $f = 1$ [40]. Informalmente, o problema consiste em encontrar, dentre os modelos de f, aquele que maximiza a soma dos pesos das variáveis valoradas como verdadeiro (ou

1).

Se f é convertida para CNF, o problema é facilmente reduzido a PWMaxSAT. Basta classificar as cláusulas de f como *hard* e anexar à fórmula a cláusula unitária $(x_i, W(x_i))$, para todo $x_i \in \mathscr{X}$.

O problema PWMaxSAT, quando restrito à funções de ponderação $W : f \to \mathbb{N}$ que têm imagem no conjunto $\mathbb{N}$ dos naturais, também é polinomialmente redutível ao problema PCB. A redução consiste em, para cada cláusula *soft* $(C_i, W(C_i))$, criar uma nova variável booleana y_i e anexar à f_h a subfórmula $y_i \leftrightarrow C_i$, onde o operador $\leftrightarrow$ indica equivalência booleana [1]. A redução consiste em associar um peso nulo a todas as variáveis da fórmula original e associar o peso da cláusula C_i à variável y_i.

A técnica de converter f para CNF e utilizar um procedimento idêntico ao DPLL com ramificação e poda apresentado é comumente usada para resolver o problema PCB [40]. Também é possível resolver o problema PCB através de um algoritmo iterativo específico [40]. Tal técnica, entretanto, também requer que a fórmula seja convertida para o formato clausal e utiliza o procedimento DPLL como um módulo, o que torna seu desempenho não melhor que as técnicas apresentadas.

Outro problema relacionado é o problema de satisfabilidade com preferências [17], ou SAT-Pr. O problema consiste em, dada uma *relação de ordem parcial* de literais, encontrar o modelo *preferido* de uma fórmula f. A ordenação, ou a relação de *preferência qualitativa de literais* (ou *pql*) pode ser descrita por um par $< S, \prec >$, onde S é um conjunto de literais e $\prec$ é uma relação sobre S. A relação $x_1 \prec \neg x_2$, por exemplo, indica que é preferido tornar o literal x_1 verdadeiro ao literal $\neg x_2$, com $x_1, \neg x_2 \in S$. A relação de preferência entre modelos pode ser obtida através de uma extensão da relação de ordenação dada entre os literais [17].

O problema PCB (e, consequentemente, o problema MaxSAT restrito à funções de ponderação em $\mathbb{N}$) pode ser reduzido para SAT-Pr. Para definir tal redução, seja $n = \lceil \lg(\sum_{x_i \in \mathscr{X}} W(x_i)+1) \rceil$ a quantidade de *bits* necessária para representar, em binário, a soma dos pesos das variáveis presentes em uma fórmula f. Além disso, seja $S' = \{b_{n-1}, ..., b_0\}$

[1] $a \leftrightarrow b = (a \to b) \wedge (b \to a) = (\neg a \vee b) \wedge (\neg b \vee a)$

um conjunto de variáveis booleanas, disjunto a $\mathscr{X}$.

Seja $Somador(S, W)$ uma fórmula que representa um somador que acumula os pesos das variáveis verdadeiras de acordo com uma valoração A. Nesta fórmula, a saída do somador é dado pelas variáveis $b_{n-1}, ..., b_0$, que representam a soma em binário, do *bit* mais significativo (b_{n-1}) ao menos (b_0). O capítulo 6 apresenta uma construção possível de uma fórmula com este comportamento.

Considere também a relação $\prec$ definida por $\{b_i \prec b_j : 0 \leq j < i < n\}$.

Os modelos preferidos de $f' = f \cup Somador(S, W)$, com relação a $< S', \prec >$ são também modelos ótimos da instância original de PCB, e a valoração das variáveis em S' pode ser utilizada para se recuperar o valor da soma máxima requerida. Intuitivamente, a pql $< S', \prec >$ indica a preferência por atribuir o valor 1 nos *bits* mais significativos da soma, fazendo com que somas maiores sejam preferíveis a somas menores. Assim, o modelo preferido é aquele que, de fato, maximiza o valor da função objetivo. Com tal redução, os algoritmos utilizados na resolução do problema SAT-Pr podem ser utilizados para se resolver PCB e, da mesma forma, PWMaxSAT restrito.

O problema SAT-Pr pode ser resolvido com uma variação do procedimento DPLL, denominada OPT-DLL [17]. A diferença entre os procedimentos está no fato de que no algoritmo OPT-DLL, a cada passo, um literal x_i é decidido (ao invés de uma variável), e x_i é propagado na fórmula. Em caso da propagação resultar em uma fórmula insatisfatível, o valor oposto do literal escolhido $\neg x_i$ é propagado. Diferentemente do processo DPLL com ramificação e poda, caso um modelo para a fórmula seja encontrado, o algoritmo para e imediatamente retorna tal valoração. Este modelo é garantidamente ótimo caso a decisão de literais seja feita pelas seguintes regras [17]:

- Se existe um literal $x_i \in S$ e todos os literais $x_j \in S$ tais que $x_j \prec x_i$ já foram valorados, então x_i é escolhido;
- Caso contrário, o literal pode ser escolhido heuristicamente como no procedimento DPLL.

Além disso, se a fórmula de entrada for convertida para seu formato clausal antes do

início do algoritmo, a regra de cláusula unitária também pode ser utilizada [17].

4.3 Considerações

Este capítulo apresenta algoritmos conhecidos para a resolução dos problemas SAT e MaxSAT. As técnicas apresentadas no capítulo 5 são modificações de um resolvedor SAT não clausal que utiliza técnicas apresentadas nesse capítulo.

CAPÍTULO 5

MODIFICAÇÕES NO RESOLVEDOR LIAMFSAT

Este capítulo apresenta as modificações realizadas em um resolvedor SAT para o tornar capaz de resolver instâncias de MaxSAT geradas através das reduções apresentadas pelo capítulo 3, além das adições realizadas no mesmo para melhorar seu desempenho.

Tais alterações consistem na modificação do resolvedor LIAMFSAT [41] para que o mesmo resolva instâncias SAT e MaxSAT eficientemente. Este resolvedor foi escolhido por ter sido desenvolvido no mesmo grupo de pesquisa deste trabalho. Além disso, seu código fonte foi gentilmente cedido para os autores deste trabalho, e algum suporte foi disponibilizado pelo autor do resolvedor. Desta forma, as modificações realizadas no código do mesmo foram feitas agilmente.

O conjunto de alterações consiste em mudanças essenciais (sem as quais o LIAMFSAT não seria capaz de resolver as instâncias geradas), e do acréscimo de dois módulos externos ao resolvedor (de Análise e de Inferência) que visam melhorar seu desempenho.

5.1 Modificações essenciais

É válido citar, como apresentado no capítulo 4, que o LIAMFSAT é um resolvedor SAT não clausal, implementado em C e $C++$, que utiliza o DPLL como procedimento base, e o ISCAS como formato de representação da fórmula de entrada.

Foi necessário modificar o formato ISCAS para suportar uma representação das instâncias geradas pelas reduções apresentadas e, naturalmente, tornar o LIAMFSAT um resolvedor MaxSAT.

5.1.1 Modificações no formato ISCAS

Primeiramente, o *parser* utilizado para a leitura de entrada no formato ISCAS foi alterado para que o operador Escolhe-H possa ser utilizado. Em particular, foi adicionada

à sintaxe a expressão

```
f = CHOOSE(n, f1, f2, ..., fk)
```

para representar o operador $C_n(f_1, ..., f_k)$, e a expressão

```
f = CHOOSE02(f1, f2, ..., fk)
```

para representar o operador $C_{\{0,2\}}(f_1, .., f_k)$. Nota-se que o *parser* modificado não é genérico o bastante para poder codificar todas as instâncias possíveis do operador Escolhe-H, mas apenas aquelas utilizadas pelas reduções apresentadas. O uso exclusivo destes operadores permite melhorias no desempenho do resolvedor.

Assim, os vértices da árvore que representa uma fórmula podem ser classificados, além dos tipos já presentes no formato ISCAS original, em *Choose* e *Choose02*. Um novo campo inteiro foi adicionado a cada nó para armazenar o valor de n nos operadores C_n.

Além disso, tal formato também foi modificado para que instâncias MaxSAT possam ser representadas. A raiz da fórmula *hard* é representada pela expressão

```
OUTPUT(f1)
```

como o formato já previa. O conjunto de fórmulas *soft*, por sua vez, é representado com as seguintes novas expressões:

```
SOFT(f1, W(f1))
SOFT(f2, W(f2))
...
```

Cada vértice da árvore gerada contém um novo campo real contendo o peso da fórmula representada pelo mesmo. A figura 5.1 exemplifica a árvore que representa uma dada fórmula. O valor **x** (ou -1.0 na implementação) indica subfórmulas sem peso definido, isto é, subfórmulas que não são classificadas como *soft*. Nota-se que as fórmulas *soft* não estão ligadas à raiz da fórmula *hard*, embora é possível tornar *soft* subfórmulas da fórmula *hard*.

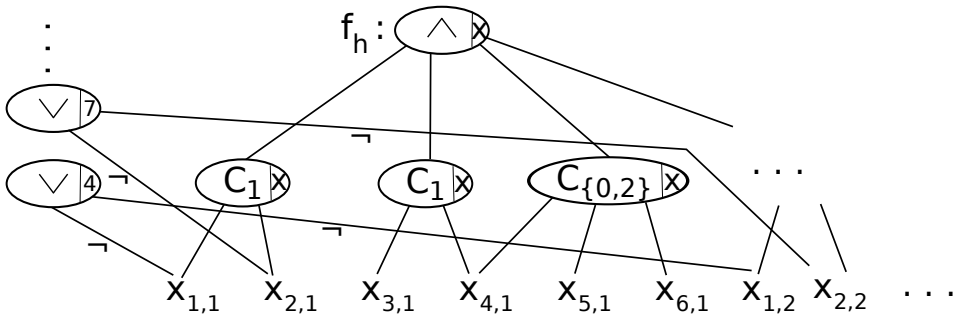

Figura 5.1: O formato ISCAS alterado para representar instâncias MaxSAT

5.1.2 Modificações na função BCP() do LIAMFSAT

O procedimento de propagação de literais ($BCP()$) do LIAMFSAT foi alterado para que os operadores C_n e $C_{\{0,2\}}$, não presentes no formato ISCAS original, possam ser utilizados.

Da mesma forma que os demais operadores, quando um vértice correspondente a um operador C_n é empilhado pela função de propagação, é verificado se seu valor verdade pode ser inferido a partir do valor associado a seus filhos no DAG. O valor verdade 0 é inferido se há mais de n filhos valorados com valor verdadeiro, ou se há mais de $k - n$ filhos valorados com valor falso, sendo k o número de filhos do vértice. O valor verdade 1 é inferido apenas se todos os filhos estão valorados, e exatamente n deles têm valor verdade 1 associado.

A inferência do valor verdade de um vértice correspondente a um operador $C_{\{0,2\}}$ ocorre de maneira análoga. O valor verdade 0 é inferido se há mais de duas subfórmulas valoradas como verdadeiro, enquanto o valor 1 é inferido se todos os seus filhos no DAG estão valorados e há exatamente 0 ou 2 valorados com o valor 1.

Se algum valor verdade é inferido, é associado tal valor ao vértice correspondente ao operador, que é então empilhado para continuar o processo de propagação.

5.1.3 O LIAMFSAT como um resolvedor MaxSAT

O LIAMFSAT foi adaptado para resolver instâncias MaxSAT pelo método de Ramificação e Poda, descrito no capítulo 4.

Tal adaptação é intuitiva. Quando o resolvedor encontra um modelo para f_h, o valor da cota superior é atualizado com o valor da cota inferior atual, o modelo é armazenado para consulta futura, e, para que o procedimento do resolvedor não pare, "finge-se" que o modelo encontrado não satisfaz a fórmula *hard*. Embora isso faça com que o LIAMFSAT sempre retorne *Não satisfatível*, o modelo ótimo de f_h pode ser impresso como resultado.

O cálculo da cota inferior é dado pela soma das fórmulas *soft* com valor verdade 0. Embora teoricamente seja um valor local de cada passo do algoritmo, o valor da cota inferior é mantido globalmente pelo programa. Tal valor, inicializado com 0, pode ser facilmente obtido durante a propagação da valoração atual na fórmula. Quando um vértice do DAG considerado *soft* é valorado com o valor 0, a cota inferior é incrementada em seu peso e, durante o processo de retrocesso, tal incremento é desfeito.

Quando a cota inferior se torna maior ou igual à cota superior em um dado passo do algoritmo, o processo de decisão de variáveis é interrompido e o resolvedor faz retrocesso.

O procedimento principal do LIAMFSAT não foi alterado. Assim, ainda pode ocorrer, durante a busca por modelos da fórmula *hard*, aprendizado de cláusulas e retrocesso não cronológico.

5.2 O Módulo de Análise

O resolvedor LIAMFSAT, com as modificações apresentadas na seção anterior, é capaz de resolver todas as instâncias obtidas através das reduções apresentadas no capítulo 3. Entretanto, é possível melhorar seu desempenho através da adaptação de técnicas que podem ser utilizadas nos problemas da Teoria de Grafos originais.

Durante a resolução dos problemas de interesse apresentados, em particular o da Árvore de Steiner e o de Ciclo Hamiltoniano Mínimo, um retrocesso pode ser realizado de forma antecipada pelo resolvedor LIAMFSAT se a valoração atual A e, principalmente, o subgrafo $G_c(A)$ correspondente forem analisados.

Para tal, um *Módulo de Análise* (MA) foi adicionado ao resolvedor. Este módulo, que é previamente alimentado com o grafo original e a relação entre suas arestas e as variáveis booleanas correspondentes, é responsável por manter o subgrafo $G_c(A)$ para

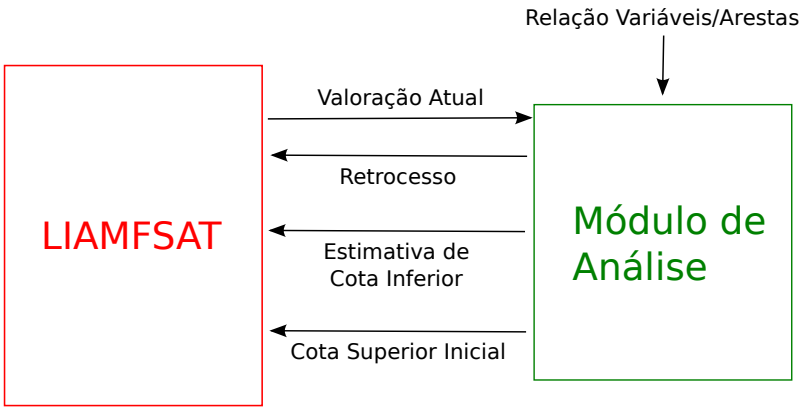

Figura 5.2: Diagrama de fluxo de dados entre o LIAMFSAT e o Módulo de Análise

toda valoração atual A do LIAMFSAT, analisar tal grafo e informar antecipadamente o resolvedor de possíveis retrocessos ou incrementos da cota inferior.

A figura 5.2 ilustra a iteração do módulo com o LIAMFSAT. Quando uma variável é decidida pelo resolvedor, o módulo de análise é informado. Dessa forma, o MA, em qualquer instante do procedimento, possui a mesma valoração parcial do LIAMFSAT.

Como o MA conhece a relação entre os literais presentes na valoração e o grafo original, a ele é possível construir o subgrafo $G_c(A)$ representado pela valoração parcial atual A. Quando o resolvedor decide ou infere o valor verdadeiro para a varável $x_{i,j}$, o índice j da variável é ignorado pelo módulo de análise e a aresta e_i é adicionada ao subgrafo mantido. Se esta aresta já estava presente em $G_c(A)$, tal valoração é ignorada. Este teste é feito em tempo constante.

Através da análise do subgrafo, é possível determinar se um corte no espaço de busca do resolvedor pode ser realizado, e informá-lo em caso positivo. Embora não discutido neste trabalho, o processo ocorre de forma análoga ao método *preguiçoso* utilizado por resolvedores de Satisfabilidade Módulo Teoria [8].

Particularmente durante a resolução do problema da Árvore de Steiner, um retrocesso pode ser realizado sempre que o grafo $G_c(A)$ contiver um ciclo. A redução do problema da Árvore de Steiner para MaxSAT apresentada no capítulo 3 garante que a solução buscada é acíclica através da maximização da função objetivo do problema. A fórmula *hard* não contém informação suficiente para fazer com que ciclos sejam interrompidos pelo

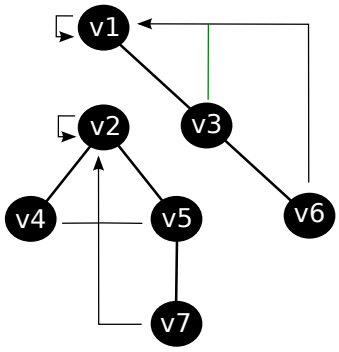

Figura 5.3: Estrutura de busca e união utilizada para detecção de ciclos.

resolvedor SAT, apenas. Assim, cabe ao módulo de análise detectar ciclos e interromper o processo de busca do LIAMFSAT.

A detecção de ciclos pelo módulo de análise pode ser feita, dentre outras maneiras, através da análise da estrutura de dados de busca e união (*Union-Find*) comumente utilizada no algoritmo de Kruskal para árvores geradoras mínimas [32].

É válido observar que, se o MA interrompe a busca do resolvedor sempre que um ciclo é encontrado, então não há ciclos representados pela valoração atual A, em qualquer passo do processo do resolvedor. Dessa forma, a cada passo, o subgrafo atual é composto por uma floresta.

Cada árvore da floresta pode ser identificada unicamente por um determinado vértice, o qual lhe pertence. Seja $pai : V \rightarrow V$ uma função tal que o vértice que representa a árvore que contém v_i é o próprio vértice v_i se $pai(v_i) = v_i$, e é o que representa a árvore que contém $pai(v_i)$ caso contrário.

Dessa forma, o vértice que identifica a árvore que contém v_i é definido por $t_i = pai(pai(...pai(v_i)...))$, tal que $pai(t_i) = t_i$. Tal cálculo corresponde a um percurso por vértices através da função pai, até que o mesmo se repita. Durante este percurso, todos os vértices visitados podem ter sua função pai atualizada diretamente para t_i, de forma a otimizar consultas posteriores a esta função.

Dois vértices v_i e v_j estão na mesma árvore se e somente se $t_i = t_j$. A figura 5.3 exemplifica uma floresta com duas árvores, identificadas pelos vértices v_1 e v_2. Flechas indicam a função $pai()$.

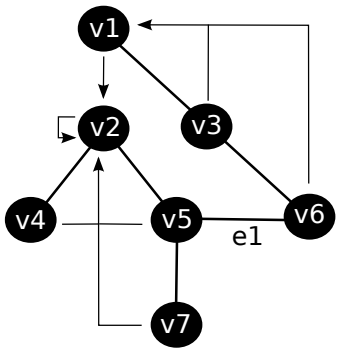

Figura 5.4: Estrutura de busca e união atualizada se a aresta $e1$ é adicionada

Quando uma aresta $e_i = \{v_i, v_j\}$ é valorada como verdadeira (ou seja, é inserida no subgrafo) através da decisão de alguma variável $x_{i,j}$, verifica-se se os vértices v_i e v_j estão na mesma árvore da floresta, o que causaria a formação de um ciclo.

Se um ciclo é formado, o resolvedor é informado e a valoração parcial atual é invalidada, causando um retrocesso. Caso contrário, a árvore de v_j passa a ser identificada pelo vértice da árvore v_i, de forma que as duas árvores passam a ser consideradas como uma árvore única. Isto pode ser feito atualizando-se a função *pai* com $pai(t_i) = t_j$ em tempo constante.

A figura 5.4 exemplifica a mudança na estrutura caso a aresta $e_1 = \{v_5, v_6\}$ seja adicionada.

Quando o resolvedor LIAMFSAT faz um retrocesso, verifica-se se a variável cujo valor verdade se torna indefinido corresponde a uma aresta e_i inserida imediatamente na última decisão realizada. Em caso negativo, tal retrocesso é ignorado. Em caso positivo, a função *pai* deve voltar a se comportar da mesma maneira de quando e_i não estava no subgrafo.

Para manter tal função de maneira eficiente, é mantida uma *pilha* de vetores onde cada vetor empilhado corresponde à descrição da função *pai* em um dado nível do resolvedor SAT.

A pilha inicia-se com a função $pai(v_i) = v_i$, para todo $v_i \in V(G)$. Quando uma nova aresta é adicionada ao subgrafo, o vetor que está no topo da pilha é duplicado, reempilhado, e então modificado como descrito acima. Tal operação tem um custo de $O(|V(G)|)$, linear no número de vértices do grafo original.

Quando uma aresta deve ser removida, entretanto, o vetor que está no topo da pilha

é simplesmente desempilhado e ignorado, em tempo constante.

Além de informar o resolvedor sempre que um retrocesso pode ser realizado, o módulo de análise também pode auxiliar no cálculo da cota inferior em um dado passo do algoritmo.

Considere novamente a floresta formada em um dado instante do processo em $G_c(A)$. Sejam s_i e s_j vértices de S e sejam T_i e T_j as árvores que os contém, respectivamente. Se $T_i \neq T_j$, para garantir a conexidade do subgrafo procurado, um custo adicional de no mínimo m unidades pode ser incrementado à cota inferior atual, onde m é o peso do menor caminho mínimo entre um vértice de T_i e um vértice de T_j. Dessa forma, a cota inferior pode ser seguramente incrementada em m unidades.

Iterar em cada par de árvores T_i e T_j possíveis e utilizar um algoritmo de caminho mínimo, entretanto, é computacionalmente caro para ser realizado regularmente. Assim, o módulo de análise calcula o caminho mínimo apenas entre $min(|S| - 1, 5)$ vértices de S e a árvore que contém o primeiro vértice de S dado na entrada.

O módulo utiliza o algoritmo de Dijkstra [18] em $G_c(A)$ para o cálculo de tais caminhos. Tal algoritmo, entretanto, não é computacionalmente barato o suficiente para que possa ser executado a cada passo do resolvedor LIAMFSAT. Assim, tal incremento na cota inferior é calculado a cada 100 iterações, apenas.

Também é possível incrementar o valor da cota inferior através da observação de que, como os vértices de S devem estar conectados na solução buscada, as árvores de $G_c(A)$ que contém algum vértice de S devem ser conectadas.

Seja n o número de árvores que contém algum vértice de S. Para conectá-las, é necessário o uso de, no mínimo, $n-1$ arestas ainda não valoradas. Isto se deve ao fato de que o grafo buscado na solução pode ser descrito, no mínimo, como uma árvore de $n-1$ arestas cujos vértices são as n árvores de $G_c(A)$ que contém algum vértice em S. A figura 5.5 exemplifica a situação. Nela, são necessários mais, no mínimo, $n - 1 = 2$ arestas para conectá-las.

Assim, a cota inferior pode ser incrementada em M unidades, onde M é igual à soma dos pesos das $n - 1$ menores arestas ainda não valoradas pela valoração atual A.

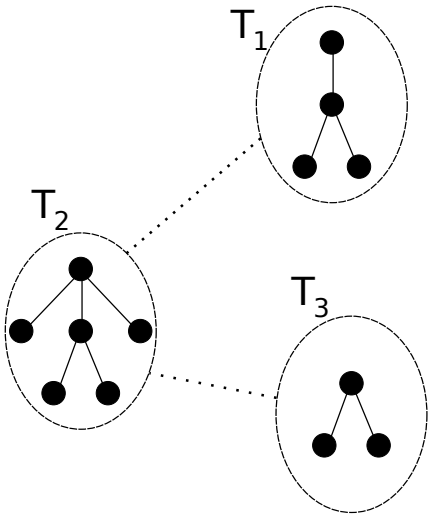

Figura 5.5: Exemplo de um subgrafo $G_c(A)$ com $n = 3$ árvores.

É possível calcular M em tempo logaritmico sobre $|E(G)|$ utilizando duas *Árvores de Fenwick* (também conhecidas como *Binary Indexed Tree* ou BIT) [19]. Dado um vetor de reais R, tal árvore permite que a operação de atualização $R[i] = y$ e a operação de consulta $\sum_{j=1}^{i} R[j]$ sejam realizadas ambas em $O(\log |R|)$.

Sejam $e_1, e_2, ..., e_{|E(G)|}$, nessa ordem, as arestas de $E(G)$ em ordem não decrescente de seus pesos. Uma BIT R é mantida de forma que $R[i]$ contém o peso da aresta e_i caso ela ainda não foi utilizada, e 0 caso contrário. Uma segunda BIT R' também é mantida, de forma que $R'[i]$ contém o valor 1 se e_i não foi utilizada, e 0 caso contrário.

A cada decisão ou implicação da aresta e_i realizada pelo resolvedor, o módulo de análise atualiza ambas as BITs na posição i, como descrito.

Para calcular o incremento desejado da cota inferior, o módulo obtém, em tempo logaritmo, o menor índice j tal que $\sum_{i=1}^{j} R'[i] = n - 1$. O índice j indica que existem exatamente $n - 1$ arestas ainda não valoradas entre e_1 e e_j, inclusive. O módulo obtém então o valor de $M = \sum_{i=1}^{j} R[i]$. Tal valor é igual à soma dos pesos das $n - 1$ menores arestas ainda não utilizadas.

Todas as operações sobre BITs custam $O(\log |E(G)|)$ operações, o que permite que tal cálculo possa ser realizado a cada decisãoou retrocesso informado para o módulo de análise.

Por fim, a cota inferior utilizada é dada pelo máximo entre a obtida com o algoritmo de

caminhos mínimos (quando executado) e com a soma dos pesos de arestas não valoradas.

A cota superior inicial é dada pelo mínimo entre três valores. O primeiro deles é o resultado da execução do algoritmo GSAT [42] na fórmula *hard*, de forma idêntica ao realizado pelo MiniMaxSAT [24].

O segundo é dado pelo custo da árvore geradora mínima do grafo original, calculado pelo módulo de análise.

O terceiro valor é dado pela soma dos custos dos caminhos mínimos entre os vértices s_1 e s_2, s_2 e s_3, ..., $s_{|S|-1}$ e $s_{|S|}$, onde $\{s_1, ..., s_{|S|}\} = S$.

A maioria das técnicas utilizadas pelo módulo de análise pode ser também utilizada durante a resolução de instâncias obtidas da redução do problema do Ciclo Hamiltoniano Mínimo.

A busca também deve sofrer retrocesso sempre que um ciclo é detectado (com a pequena diferença que tal ciclo deve ter tamanho $k < |V|$).

A cota inferior também pode ser incrementada pelo algoritmo de cálculo de caminhos mínimos descrito anteriormente ou pela soma das $n - 1$ arestas ainda não utilizadas, pois a solução buscada também deve ser conexa.

Além disso, a cota superior inicial pode ser dada pelo resultado da execução do GSAT, mas não pela árvore geradora mínima do grafo. Ao invés disso e da soma de custos de caminho, é utilizada simplesmente a soma dos pesos de todas as arestas do grafo original.

Deve-se observar que, como o problema Clique é reduzido para SAT, técnicas utilizadas em resolvedores MaxSAT, como cotas superior e inferior, não são aplicáveis a estas instâncias. Assim, o módulo de análise não é utilizado para tal problema.

5.3 O Módulo de Inferências

Embora o uso do Módulo de Análise pode melhorar o desempenho do resolvedor como observado através dos resultados apresentados no capítulo 6, um segundo módulo foi desenvolvido com o objetivo de torná-lo ainda mais eficiente.

Este módulo não utiliza informações provenientes dos problemas em grafos originais. De fato, o módulo pode ser utilizado pelo LIAMFSAT durante a resolução de qualquer

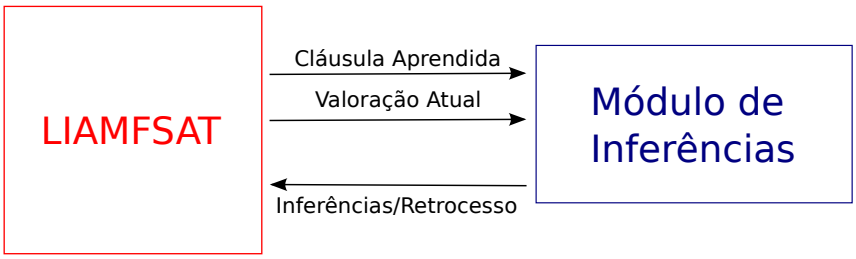

Figura 5.6: Diagrama de fluxo de dados entre o LIAMFSAT e o Módulo de Inferências

instância SAT ou MaxSAT não clausal, e não apenas durante a resolução dos problemas apresentados. Assim, tal módulo pode ser visto como uma colaboração ao LIAMFSAT, visando apenas uma possível melhora em seu desempenho.

Como apresentado no capítulo 4, o LIAMFSAT anexa as cláusulas aprendidas durante o processo à fórmula (*hard*) através de uma conjunção com a mesma. Embora estas cláusulas representem uma subfórmula em CNF, o resolvedor originalmente utiliza as mesmas técnicas aplicadas sobre a fórmula não clausal nelas. Dessa forma, o resolvedor não aplica a regra da cláusula unitária e não realiza possíveis inferências através dela.

Foi introduzido ao LIAMFSAT o *Módulo de Inferências* (MI). Este módulo é responsável por armazenar o conjunto de cláusulas aprendidas pelo LIAMFSAT e aplicar a regra da cláusula unitária, quando possível.

A figura 5.6 ilustra a iteração do resolvedor com o MI. O LIAMFSAT foi modificado de forma que, quando aprende uma cláusula, alimenta o módulo de inferências, ao invés de modificar a fórmula de entrada. Além disso, quando uma decisão do valor de uma variável é feita (ou desfeita através de retrocesso), o módulo de inferência é informado.

Assim, o MI sempre possui a mesma valoração parcial utilizada pelo LIAMFSAT, o que lhe permite propagar tal valoração nas cláusulas armazenadas e realizar inferências, se possível. Tais inferências são então informadas ao LIAMFSAT, que as realiza.

O módulo de inferência poderia armazenar o conjunto de cláusulas aprendidas pelo LIAMFSAT da mesma forma que um resolvedor SAT clausal, e realizar inferências, por exemplo, através da observação das cláusulas pela técnica de literais vigiados, apresentada no capítulo 4.

Entretanto, uma outra estrutura é utilizada para realizar tal função. Essa estrutura é

mantida de forma que apenas as cláusulas que contém um literal decidido são analisadas (como no método de literais vigiados), e de forma que uma inferência pode ser realizada em tempo constante após as análises. Além disso, tal estrutura suporta a operação de *subjugação*, na qual uma cláusula que contém inteiramente alguma outra pode ser removida da coleção. Esta operação, entretanto, não é sempre barata computacionalmente, como descrito a seguir. Nenhum resolvedor SAT no estado-da-arte implementa tal operação devido ao seu custo.

Para o entendimento dessa estrutura, nota-se que uma cláusula pode ser descrita como um conjunto de literais, e um literal pode ser mapeado para um número natural único. O módulo de inferência deve então ser capaz de armazenar uma coleção de conjuntos de números naturais.

O mapeamento de literais para números ocorre de forma que o literal x_i é mapeado para o número $num(x_i) = 2 \times i + sig(x_i)$. Dessa forma, os literais $\neg x_0, x_0, \neg x_1, x_1, ...$ são mapeados, nessa ordem, para $0, 1, 2, 3, ...$ Uma cláusula $\{x_0, x_1, x_2, ...\}$ é então descrita pelo conjunto $\{num(x_0), num(x_1), num(x_2), ...\}$.

Para que o método de inferência do MI seja facilitado, o LIAMFSAT, quando aprende uma cláusula C, alimenta o módulo com o conjunto das negações dos literais que compõem C. Dessa forma, se o resolvedor aprender, por exemplo, a cláusula $\{x_0, \neg x_1, x_3, \neg x_4\}$, o módulo de inferência será alimentado com a cláusula $\{\neg x_0, x_1, \neg x_3, x_4\}$, que será então mapeada pelo MI para o conjunto $\{0, 3, 6, 9\}$.

A estrutura de dados utilizada para armazenar os conjuntos recebidos pelo resolvedor é uma versão da *SetTrie* [10] otimizada para a realização de inferências. A principal ideia dessa estrutura está no fato de que cada conjunto de números, quando ordenado de acordo com a relação em $\mathbb{N}$, pode ser tratado como uma *string*. Dessa forma, os conjuntos podem ser armazenados em uma árvore de prefixos, ou *trie*. A figura 5.7 exemplifica uma *SetTrie* contendo os conjuntos $\{0, 3, 6, 9\}, \{0, 13, 11, 3, 14\}$ e $\{6, 9\}$. Nós pretos indicam o final de um conjunto.

O alfabeto $\sum = \{0, 1, ..., 2 \times |\mathscr{X}| - 1\}$ necessário para representar os conjuntos como *strings* tem tamanho $|\sum| = 2 \times |\mathscr{X}|$. Armazenar um vetor de tal tamanho em cada nó da

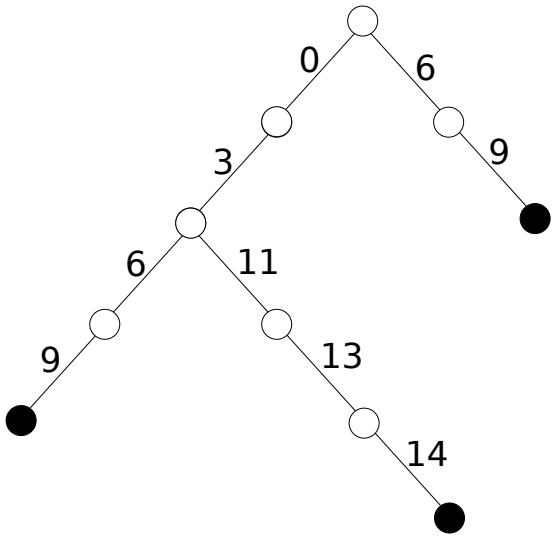

Figura 5.7: *SetTrie* armazenando os conjuntos $\{0, 3, 6, 9\}, \{0, 13, 11, 3, 14\}$ e $\{6, 9\}$.

árvore é inviável para instâncias grandes. Por este motivo, cada nó da árvore contém uma árvore rubro-negra (um *set* provido pela *Standard Template Library* (STL) da linguagem C++) para armazenar sua lista de filhos. Assim, para cada nó, há um custo logaritmo no número de filhos do mesmo para se percorrer a árvore.

Como descrito, o módulo de inferência é alimentado com as valorações decididas pelo LIAMFSAT, de forma que, em qualquer passo do procedimento, a valoração parcial atual A também é totalmente conhecida pelo MI. Como apresentada no capítulo 4, a regra da cláusula unitária é aplicada quando todos seus literais são valorados com o valor falso, exceto por um, que pode ser inferido. Como o MI armazena a negação dos literais das cláusulas aprendidas, a regra de inferência pode ser utilizada quando todos os literais de um conjunto estão em A, exceto por exatamente um.

Para que este literal seja encontrado eficientemente, um valor (dito *cache*) do conjunto $\sum \cup \{TA, MU\}$ é associado a cada nó da árvore, e indica, dada a valoração atual A, o número do único literal que não está presente em A no caminho entre o nó e a raiz da árvore.

A *cache* de um nó é igual a TA (*Todos Acima*) se todos os literais do caminho estão na valoração, e igual a MU (*Mais de Um*) se há mais de um literal do caminho ausente de A. A figura 5.8 exemplifica o valor da *cache* de cada nó da árvore caso a valoração

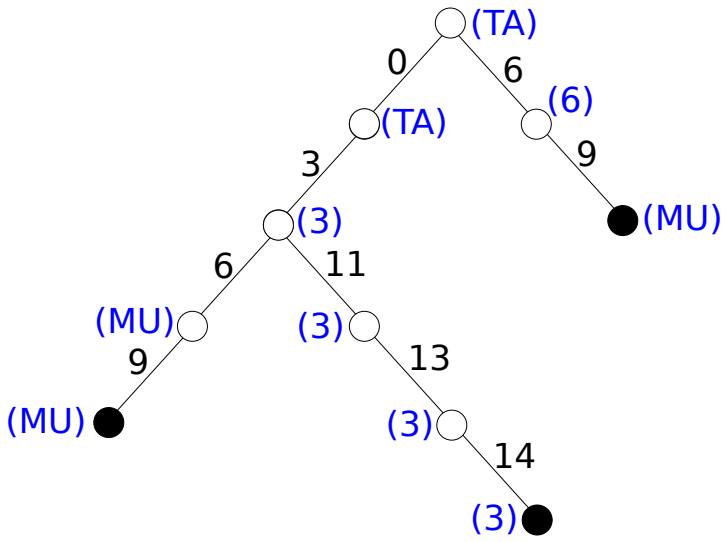

Figura 5.8: Indicação da *cache* de cada nó de uma *SetTrie* para a valoração parcial $A = \{0, 11, 13, 14\}$

atual seja $\{\neg x_0, x_5, x_6, \neg x_7\}$, associada a $\{0, 11, 13, 14\}$.

Um literal pode ser inferido sempre que houver um nó indicando fim de conjunto cuja *cache* é diferente de TA e MU. Dessa forma, o literal 3 pode ser inferido na situação de exemplo, pois é a *cache* do nó de fim do conjunto $\{0, 13, 11, 3, 14\}$. Quando isto ocorre, o literal é restaurado a partir de seu número ($num^{-1}(3) = x_1$) e o LIAMFSAT é alimentado com sua negação, realizando-se assim uma inferência.

Se houver um nó indicando fim de conjunto cuja *cache* seja TA em algum momento, um conflito é identificado, pois a valoração atual contém todas as negações de literais de uma dada cláusula. Neste caso, o LIAMFSAT é alimentado com um sinal indicando que um conflito foi encontrado e um retrocesso deve ser realizado.

A manutenção das *caches* deve ser feita de maneira eficiente. Quando o módulo de inferências é alimentado, um novo conjunto é inserido na coleção de maneira análoga à realizada em uma *trie* comum, e *caches* de nós que já estavam presentes na árvore não são alterados.

Para cada novo nó criado, sua *cache* recebe o valor TA se e somente se a *cache* de seu pai tem valor TA e o rótulo da nova aresta é um número presente na valoração atual. Se a *cache* de seu pai tem valor TA, mas o número não faz parte da valoração atual, tal número é utilizado como *cache* do novo vértice. Se a *cache* de seu pai é diferente de TA

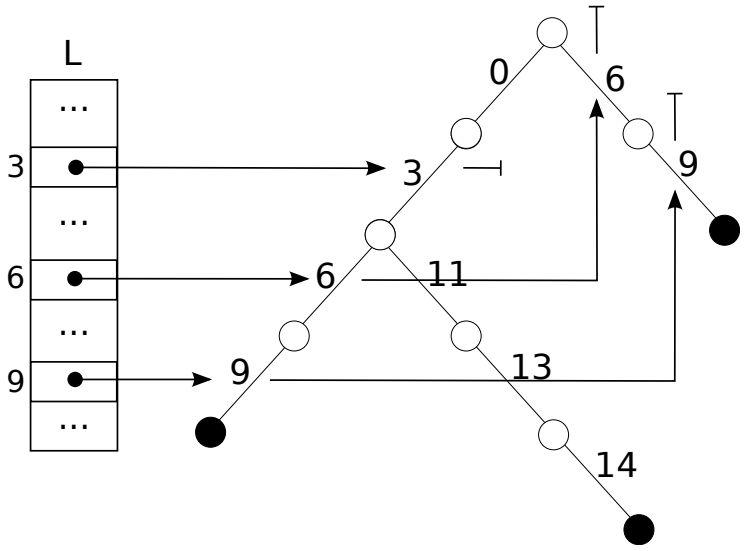

Figura 5.9: Vetor L indicando listas ligadas com referências às arestas de mesmo rótulo.

e de MU, a *cache* do novo vértice é valorada com a *cache* de seu pai caso o rótulo da nova aresta seja um número presente na valoração atual, e com MU caso contrário. Em qualquer outro caso, a *cache* recebe o valor MU.

Uma tabela é mantida de forma a fazer com que uma consulta na valoração atual tenha custo constante, o que torna o custo da inserção de um novo nó também constante.

Quando o módulo de inferências é alimentado com uma mudança na valoração atual, as *caches* devem ser alteradas de forma a serem compatíveis com a nova valoração. Tal mudança pode consistir na inserção ou na remoção de um literal x_i na valoração atual.

Em ambos os casos, após serem definidos, os nós cujas *caches* devem ser alteradas podem ser atualizados com a mesma regra utilizada no procedimento de inserção. Para que tais nós sejam encontrados eficientemente, é mantido um vetor L, de tamanho $|L| = |\sum|$, cuja posição $L[i]$ indica uma lista ligada que contém referências às arestas cujos rótulos são i. A figura 5.9 exemplifica a estrutura. Embora algumas listas foram omitidas na figura, elas existem para todos os elementos de L.

A manutenção das *caches* é realizada percorrendo-se a lista $L[num(x_i)]$. Para cada aresta e da lista, nenhum nó acima de e terá o valor de sua *cache* alterado. Desta forma, basta atualizar o valor da *cache* para o nó pertencente à aresta, imediatamente abaixo na árvore. Se (e somente se) a *cache* desse nó for alterado, o procedimento é chamado

recursivamente para as arestas que ligam tal nó a todos os seus filhos.

Isto é suficiente para manter todas as *caches* compatíveis com a valoração atual. Além disso, quando uma nova aresta é criada pelo procedimento de inserção, sua respectiva lista em L é atualizada em tempo constante, uma vez que basta inseri-la em uma lista ligada simples.

Como citado, esta estrutura permite a operação de subjugação. Quando um novo conjunto é inserido na árvore, todos os seus superconjuntos próprios podem ser removidos da mesma. Para tal, basta listá-los através de um percurso específico na árvore [10], e removê-los como em uma *trie* comum. A manutenção das *caches* e das listas em L se dá de forma idêntica à feita no procedimento de inserção. Na árvore de exemplo, o conjunto $\{0, 3, 6, 9\}$ seria removido quando o conjunto $\{6, 9\}$ fosse inserido.

Tal percurso, no entanto, potencialmente visita toda a árvore de prefixos, o que o pode tornar inviável para uma coleção grande de cláusulas armazenadas. Por isso, esta função não está ativada por padrão no resolvedor. O capítulo 6 apresenta os resultados de experimentos realizados com esta função ativada.

5.3.1 O Módulo de Inferências no Módulo de Análise

Além de auxiliar o processo do LIAMFSAT, o Módulo de Inferências também pode ser utilizado para sofisticar o procedimento do Módulo de Análise. Em particular, além de realizar cortes no espaço de busca quando um ciclo é detectado, o Módulo de Análise também poderia inferir o valor verdade de variáveis, de forma a impedir a formação de ciclos em iterações próximas.

Como exemplo, suponha que $G' = (\{v_1, v_2, v_3\}, \{e_1 = \{v_1, v_2\}, e_2 = \{v_2, v_3\}, e_3 = \{v_3, v_1\}\})$ é um subgrafo do grafo original, e suponha também que, em um dado instante do processo, a valoração atual faz com que as arestas e_1 e e_2 estejam presentes em $G_c(A)$. Nesse caso, é possível inferir que a aresta e_3, e portanto variáveis $x_{3,j}$, não podem estar presentes na solução procurada.

Isto pode ser feito através do *aprendizado de ciclos*. Quando um ciclo é detectado pelo método descrito anteriormente, uma busca em profundidade [14] é utilizada em $G_c(A)$

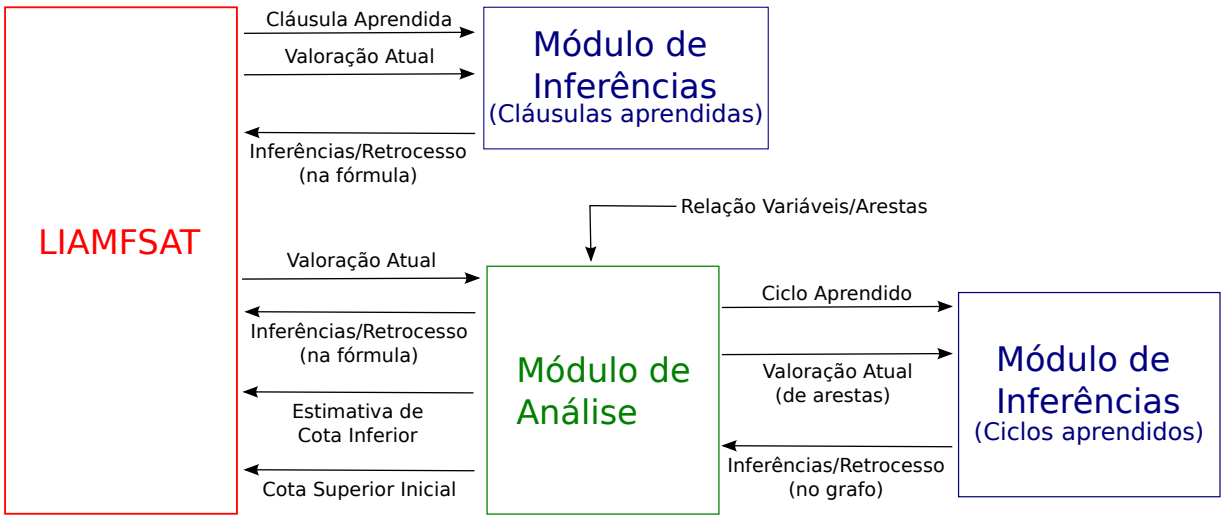

Figura 5.10: Diagrama de fluxo de dados completo

para encontrar o conjunto $\{e_1, e_2, ..., e_k\}$ de arestas que pertencem ao ciclo.

Associando-se cada aresta $e_i \in E(G)$ com uma variável booleana $\varrho_i \in \mathscr{E} = \{\varrho_i : e_i \in E(G)\}$, a cláusula $\neg(\varrho_1 \wedge \varrho_2 \wedge ... \wedge \varrho_k) = (\neg\varrho_1 \vee \neg\varrho_2 \vee ... \vee \neg\varrho_k)$ é aprendida.

Como o módulo de análise ignora o grafo ao qual uma aresta representada por uma variável pertence, o conjunto das variáveis pertencentes a essa nova cláusula não é um subconjunto das variáveis utilizadas pelo LIAMFSAT, $\mathscr{X}$, mas sim de $\mathscr{E}$.

Por isso, a cláusula não pode ser inserida no módulo de inferências utilizado diretamente pelo LIAMFSAT. Por este motivo, um segundo módulo de inferências, cuja implementação é idêntica ao primeiro, é utilizado para o armazenamento das cláusulas.

A figura 5.10 demostra o diagrama de fluxo de dados completo do resolvedor e seus módulos adicionais.

5.4 Considerações

Este capítulo apresenta as modificações realizadas no resolvedor LIAMFSAT a fim de torná-lo capaz de resolver as instâncias dos problemas reduzidos no capítulo 3. Tais modificações formam uma das principais contribuições deste trabalhado, permitindo que um resolvedor SAT não clausal seja utilizado para experimentar as reduções.

Foram experimentados o resolvedor completo, sem o Módulo de Inferências, sem o

Módulo de Análise e sem ambos os módulos. A função de subjugação do Módulo de Inferências também foi testada, assim como o Módulo de Análise sem o LIAMFSAT. O resultado de tais experimentos são apresentados no capítulo 6.

CAPÍTULO 6

RESULTADOS EXPERIMENTAIS

Este capítulo apresenta resultados de experimentos realizados para avaliar o tamanho das instâncias geradas pelas reduções apresentadas na seção 3.2, e o desempenho do resolvedor apresentado no capítulo 5 para resolver as mesmas.

O capítulo está dividido em quatro seções, onde cada uma apresenta os resultados obtidos durante a resolução dos problemas da Árvore de Steiner, do Ciclo Hamiltoniano, do Ciclo Hamiltoniano Mínimo, e de Clique.

As reduções dos problemas com soluções polinomiais conhecidas não foram experimentadas por serem apenas sub-rotinas das reduções dos problemas experimentados.

Foram utilizadas instâncias provenientes de *benchmarks* conhecidos para cada problema [31, 50], além de instâncias geradas aleatoriamente.

Para cada uma das instâncias utilizadas para cada problema, são comparados os desempenhos dos seguintes métodos para resolvê-la, quando aplicáveis:

- (MLS) A redução apresentada na seção 3.2, cuja instância obtida é resolvida pelo resolvedor apresentado no capítulo 5;
- (MLSj) A resolução da instância obtida da mesma redução, realizada pelo mesmo resolvedor, cuja função de subjugação do Módulo de Inferências é utilizada;
- (MLS$\bar{a}$) A resolução da instância obtida da mesma redução, realizada pelo mesmo resolvedor, mas sem aprendizado e retrocesso não cronológico. Isto inutiliza o módulo de inferências;
- (MLS$\bar{m}$) A resolução da instância obtida da mesma redução, realizada pelo mesmo resolvedor, mas sem o uso do módulo de análise. Assim, a instância é resolvida apenas por um maquinário para satisfabilidade;

- (MLS$\bar{a}\bar{m}$) A resolução da instância obtida da mesma redução, realizada pelo mesmo resolvedor, mas sem aprendizado, retrocesso não cronológico, e módulo de análise;

- (MA) A resolução da instância realizada apenas pelo módulo de análise. O LIAMF-SAT é substituído por um módulo semelhante que trabalha sobre uma representação de tamanho linear no tamanho do grafo original.

  Esse módulo enumera, através de retrocesso, valorações sobre o conjunto de variáveis $\mathscr{X} = \{x_i : e_i \in E(G)\}$, onde G é o grafo original da instância. De maneira semelhante ao DPLL, a cada passo do processo, uma variável de $\mathscr{X}$ é escolhida e um valor verdade é decidido.

  Esta decisão, entretanto, não é propagada em nenhuma fórmula booleana, mas é apenas informada ao Módulo de Análise. O Módulo de Análise não tem seu comportamento alterado: ele informa então o enumerador quanto a inferências, retrocessos e cotas, sempre que preciso.

  Quando o enumerador obtiver uma valoração completa, verifica-se se o subgrafo correspondente é factível para o problema (verifica-se, por exemplo, se o grafo tem todos os vértices de um conjunto S conectados, ou se é formado por um ciclo hamiltoniano), substituindo-se assim a verificação de satisfabilidade de uma fórmula *hard*.

- (CNFr) A resolução da instância obtida da mesma redução, mas convertida para SAT ou MaxSAT clausal. Tal resolução é realizada pelo resolvedor SAT clausal Lingeling [11] ou pelo resolvedor MaxSAT clausal MiniMaxSAT [24], de acordo com o problema;

- Uma redução publicada anteriormente do problema para SAT ou MaxSAT clausal, descrita na seção 3.1. A instância obtida pela redução é resolvida pelo resolvedor Lingeling ou pelo resolvedor MiniMaxSAT;

- Uma técnica que utiliza um resolvedor SAT ou MaxSAT como subrotina para resolver as instâncias originais dos problemas.

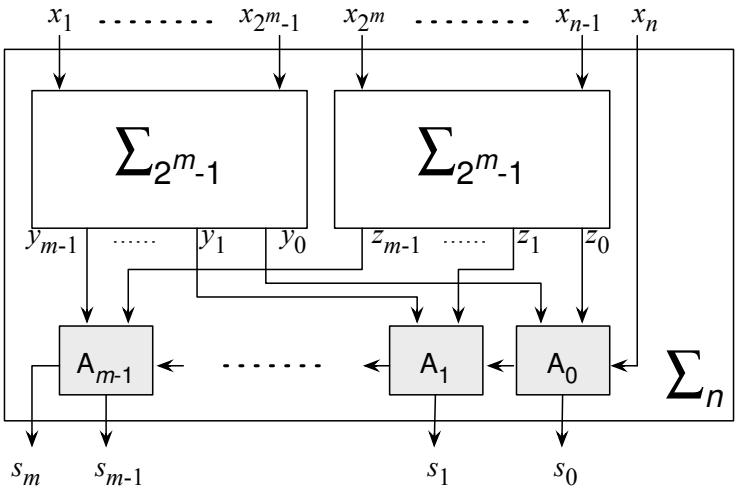

Figura 6.1: Esquema da codificação em CNF de um somador de n entradas [38].

Para que seja possível experimentar as instâncias obtidas pela redução apresentada neste trabalho convertidas para CNF, como descrito no penúltimo item acima, é necessária uma conversão eficiente do operador Escolhe-H para uma fórmula clausal.

Todos os operadores Escolhe-H são convertidos através da codificação de um circuito lógico que consiste em um contador e um comparador.

O contador pode ser convertido para CNF através da codificação recursiva de um somador paralelo [38], como ilustrado pela figura 6.1.

Para codificar um somador cuja entrada consiste nas variáveis $x_1, ..., x_n$, $n-1$ variáveis são divididas em dois grupos, de tamanhos $2^m - 1$ e $(n-1) - (2^m - 1)$, onde m é o maior natural tal que $2^m - 1 \leq \lfloor (n-1)/2 \rfloor$. Cada grupo é interpretado como a entrada de um somador semelhante, estabelecendo-se assim uma recursão. Suas saídas são então somadas com um conjunto de somadores completos (módulos A_i da figura 6.1).

Somadores simples, de até três entradas, são codificados diretamente com somadores completos e incompletos, em CNF. A codificação de um somador incompleto das variáveis a e b, por exemplo, é feita através da criação das novas variáveis booleanas s_i e s_j que representam a saída do somador, e pela codificação, em CNF, da fórmula $(s_i \leftrightarrow (a \oplus b)) \wedge (s_j \leftrightarrow (a \wedge b))$[1] [38].

[1] $a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$

Esse somador cria $O(n)$ cláusulas e até $2(n-1)$ variáveis novas [38]. Entretanto, devido à maneira com que o comparador é codificado, várias delas podem ter seu valor verdade fixado, e podem portanto ser retiradas.

O comparador, por sua vez, é codificado de acordo com o conjunto H do operador. Se H é unitário e contém apenas o natural k, a codificação pode ser realizada com um conjunto de cláusulas unitárias, contendo a representação binária de k na saída do contador.

Assim, se o contador tem como saída as variáveis $s_0, ..., s_m$, basta incluir na fórmula resultante a cláusula unitária (s_i) se o $i-$ésimo *bit* de k é 1, ou a cláusula $(\neg s_i)$ caso contrário, para todo $0 \leq i \leq m$. Em particular, para codificar o operador C_1, basta incluir as cláusulas $(s_0), (\neg s_1), (\neg s_2), ..., (\neg s_m)$.

Para se codificar o operador $C_{\{0,2\}}$, basta indicar que todos os *bits* da saída do contador devem ser 0, exceto pelo de índice 1, que pode assumir qualquer valor. Assim, basta incluir na fórmula as cláusulas unitárias $(\neg s_0), (\neg s_2), (\neg s_3), ..., (\neg s_m)$.

Todas as cláusulas unitárias de uma fórmula são removidas imediatamente por um resolvedor SAT clausal pela regra de cláusula unitária.

Outras maneiras de se converter o operador Escolhe-H para CNF foram estudadas e descartadas [2, 35, 7].

É possível codificar o operador C_k (para H unitário) utilizando uma representação unária do valor de k, ao invés de sua representação binária [2]. Entretanto, tal codificação gera uma fórmula cujo número de cláusulas é quadrático na aridade do operador, o que a torna maior do que a gerada pelo método apresentado anteriormente. Além disso, seria necessário codificar dois operadores e uma disjunção para representar o operador $C_{\{0,2\}}$.

Existe uma codificação robusta para CNF do operador $C_{\{0,1\}}$, em particular [35]. Entretanto, o operador C_2 não é tratado, de forma que seria necessário, para codificar o operador $C_{\{0,2\}}$, transformar os operadores C_2 e $\neg C_1$ com algum outro método, e ainda criar conjunções e disjunções entre eles. Isto levaria à inclusão de ainda mais variáveis novas.

O operador $C_{\{a,a+1,...,b-1,b\}}, 0 \leq a \leq b$ pode ser reescrito em CNF através de representação unária com um somador recursivo análogo ao apresentado e um comparador

específico para o operador [7]. Tal método é semelhante ao utilizado, e tal operador é muito genérico para as reduções utilizadas neste trabalho.

Por fim, também é possível descrever o operador Escolhe-H como um conjunto de equações pseudo-booleanas e utilizar um conversor delas para CNF, como o BoolVarPB [6]. Tal método foi utilizado no início deste trabalho, mas foi substituído pela técnica apresentada por esta apresentar um desempenho melhor com as instâncias obtidas.

As fórmulas *soft*, por conterem apenas operadores da lógica proposicional, podem ser convertidas através de transformações lineares conhecidas, como a de Tseitin [45]. Cada fórmula $f_{s_i} \in f_s$ é convertida para uma fórmula em CNF f'_{s_i} pela transformação. Então, uma nova variável booleana y_i é criada e são anexadas à fórmula as cláusulas *hard* que indicam a relação $y_i \leftrightarrow f'_{s_i}$. Uma cláusula *soft* unitária y_i, cujo peso é idêntico ao peso da fórmula f_{s_i} na instância original, também é adicionada.

Todos os experimentos foram realizados em uma máquina de dois núcleos *Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz* com *6 Mb* de memória *Cache* e *4 Gb* de memória RAM.

Os tempos fornecidos como resultados correspondem ao tempo de uso de CPU de cada método, e são sempre dados em segundos. Além disso, o tempo de execução de cada método foi limitado em 1800 segundos (30 minutos).

6.1 Árvore de Steiner

Esta seção apresenta os experimentos realizados com o problema da Árvore de Steiner.

Para a experimentação com este problema, foram utilizadas instâncias do *Benchmark* SteinLib [31], em particular das categorias B e I080. Ambas as categorias contém grafos aleatórios esparsos. As arestas dos grafos da categoria B são ponderadas aleatoriamente com distribuição Uniforme, enquanto as da categoria I080 são ponderadas aleatoriamente com distribuição Normal [31].

A tabela 6.1 exibe o tamanho das instâncias obtidas pelas reduções. A sigla CNFj representa a redução apresentada na subseção 3.1.1. Cada campo é apresentado na forma $|\mathscr{X}|/|f|$, onde $|\mathscr{X}|$ é o numero de variáveis utilizadas pela fórmula gerada, e $|f|$ é o número

de cláusulas da mesma. Assim, a fórmula gerada pela redução apresentada no capítulo 3 convertida para CNF, da instância *i080-001*, por exemplo, usa 2010 variáveis e contém 8285 cláusulas.

A tabela 6.2 exibe os desempenhos obtidos dos métodos comparados. As fórmulas da redução CNFj foram resolvidas pelo resolvedor MiniMaxSAT. Tempos em negrito indicam o melhor método avaliado para resolver a instância dada, quando único. A sigla TLE (*Tempo Limite Excedido*) indica que o método não resolveu a instância dada dentro do tempo limite estipulado. Todos os métodos que resolveram instâncias no tempo estipulado apresentaram respostas corretas, idênticas às publicadas pelos autores do *Benchmark*.

Tabela 6.1: Tamanhos das fórmulas obtidas pelas reduções com instâncias das categorias B e I080 da SteinLib

Instância	$\|V\|/\|E\|/\|S\|$	|CNFj|	|CNFr|
b01	50/63/9	15939/1032075	1559/6341
b02	50/63/13	15939/1032079	2211/8998
b03	50/63/25	15939/1032091	4455/18209
b04	50/100/9	40100/4080209	2804/12512
b05	50/100/13	40100/4080213	4060/18498
b06	50/100/25	40100/4080225	8356/37014
b07	75/94/13	35438/3393225	3286/13480
b08	75/94/19	35438/3393231	4810/20128
b09	75/94/38	35438/3393250	10232/41018
i080-001	80/120/6	57720/7027446	2010/8285
i080-002	80/120/6	57720/7027446	2000/8247
i080-003	80/120/6	57720/7027446	1960/8116
i080-004	80/120/6	57720/7027446	2020/8383
i080-011	80/350/6	490360/172480706	7250/36035
i080-012	80/350/6	490360/172480706	7360/36270
i080-013	80/350/6	490360/172480706	7360/36240

Primeiramente, através da análise da tabela 6.1, é possível verificar que as fórmulas convertidas da redução apresentada na subseção 3.2.4 são menores que as obtidas pela redução CNFj, tanto em número de variáveis quanto em número de cláusulas. Isto é facilmente justificado pelo tamanho assintótico das fórmulas: A redução CNFj gera fórmulas com $O(|E|^3)$ cláusulas, enquanto a fórmula gerada pela CNFr contém um número quadrático de elementos (convertidos para CNF linearmente).

Entretanto, através da análise da tabela 6.2, é possível notar que as instâncias da

Tabela 6.2: Resultados experimentais com instâncias das categorias B e I080 da SteinLib

Instância	$\|V\|/\|E\|/\|S\|$	CNFj	MLS	MLSj	MLS $\bar{a}$	MLS$\bar{m}$	MLS$\bar{a}\bar{m}$	MA	CNFr
b01	50/63/9	TLE	13.5	12.3	17.1	TLE	TLE	TLE	**0.3**
b02	50/63/13	TLE	37.1	35.3	86.5	TLE	TLE	TLE	**18.8**
b03	50/63/25	TLE	TLE	TLE	TLE	TLE	TLE	TLE	**8.7**
b04	50/100/9	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
b05	50/100/13	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
b06	50/100/25	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
b07	75/94/13	TLE	TLE	TLE	TLE	TLE	TLE	TLE	**39.8**
b08	75/94/19	TLE	TLE	TLE	TLE	TLE	TLE	TLE	**1754.0**
b09	75/94/38	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
i080-001	80/120/6	TLE	TLE	TLE	707.6	TLE	TLE	TLE	**152.6**
i080-002	80/120/6	TLE	87.7	85.3	151.8	TLE	TLE	TLE	**1.4**
i080-003	80/120/6	TLE	**34.8**	35.1	65.0	TLE	TLE	TLE	192.7
i080-004	80/120/6	TLE	333.0	323.2	TLE	TLE	TLE	TLE	**148.5**
i080-011	80/350/6	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
i080-012	80/350/6	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
i080-013	80/350/6	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE

SteinLib não são suficientemente pequenas para que a diferença entre o desempenho dos métodos CNFj, MLS$\bar{a}$, MLS$\bar{m}$, MLS$\bar{a}\bar{m}$ e MA seja perceptível.

Assim, instâncias menores foram geradas aleatoriamente para que tal diferenciação seja possível. As tabelas 6.3 e 6.4 apresentam os resultados obtidos com tais instâncias. Todos os métodos que resolveram instâncias no tempo limite apresentaram a mesma resposta para uma mesma instância.

Algumas observações podem ser feitas através da análise das tabelas apresentadas nesta seção.

Primeiramente, embora polinomial, a redução utilizada no método CNFj gera fórmulas muito grandes em relação ao tamanho original das instâncias, como observado. Assim, tais fórmulas não são resolvidas pelo MiniMaxSAT no tempo estipulado para grande parte das instâncias testadas. O método CNFr, que também consiste na redução do problema para MaxSAT clausal, consegue resolver a maioria das instâncias experimentadas.

É possível notar também que as instâncias geradas pela redução apresentada na subseção 3.2.4, quando convertidas para CNF, são geralmente resolvidas (até 180 vezes) mais rápido do que o resolvedor apresentado no capítulo 5, que trabalha com um formato não

Tabela 6.3: Tamanhos das fórmulas obtidas pelas reduções com instâncias aleatórias

Instância	$\|V\|/\|E\|/\|S\|$	|CNFj|	|CNFr|
rand01	8/14/4	798/12576	152/616
rand02	9/18/4	1314/25960	198/834
rand03	9/18/5	1314/25961	282/1160
rand04	9/18/6	1314/25962	328/1401
rand05	9/18/7	1314/25963	402/1680
rand06	9/18/8	1314/25964	480/2003
rand07	9/18/9	1314/25965	514/2220
rand08	10/23/2	2139/52948	102/411
rand09	10/23/3	2139/52949	189/804
rand10	10/23/4	2139/52950	272/1165
rand11	10/23/5	2139/52951	363/1566
rand12	11/28/5	3164/94141	396/1876
rand13	12/33/6	4389/152532	598/2871
rand14	15/35/8	4935/181378	882/3997
rand15	20/45/13	8145/380803	2001/8898
rand16	25/54/15	11718/653307	2686/11896
rand17	30/70/17	19670/1411357	4038/18300

Tabela 6.4: Resultados experimentais com instâncias aleatórias

Instância	$\|V\|/\|E\|/\|S\|$	CNFj	MLS	MLSj	MLS $\bar{a}$	MLS$\bar{m}$	MLS$\bar{a}\bar{m}$	MA	CNFr
rand01	8/14/4	1.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand02	9/18/4	28.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand03	9/18/5	38.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand04	9/18/6	149.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand05	9/18/7	32.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand06	9/18/8	23.9	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand07	9/18/9	10.3	0.1	0.1	0.1	1.3	0.3	0.1	0.1
rand08	10/23/2	6.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand09	10/23/3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand10	10/23/4	12.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
rand11	10/23/5	716.7	0.1	0.1	0.1	0.2	0.2	0.1	0.1
rand12	11/28/5	TLE	0.1	0.1	0.1	0.3	0.2	0.1	0.2
rand13	12/33/6	TLE	0.4	0.4	**0.3**	56.0	12.9	0.5	0.8
rand14	15/35/8	TLE	0.6	0.9	**0.3**	153.6	34.7	4.0	3.4
rand15	20/45/13	TLE	14.5	14.1	**4.8**	TLE	TLE	432.9	135.3
rand16	25/54/15	TLE	90.8	88.9	**53.7**	TLE	TLE	TLE	493.4
rand17	30/70/17	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE

clausal, para as instâncias da SteinLib.

Entretanto, o resolvedor não clausal tem desempenho idêntico ou (até 20 vezes) melhor

para a instância *i080-003* e para todas as instâncias aleatórias da tabela 6.4. É possível que tais instâncias contenham características específicas que tornam ruins as heurísticas utilizadas pelo MiniMaxSAT durante a resolução.

É interessante comparar os resultados dos métodos MLS$\bar{a}$ e MA. Embora ambos os métodos utilizem o módulo de análise igualmente, o método MLS$\bar{a}$ usa como procedimento base uma busca em uma representação quadrática no tamanho do grafo original, enquanto o MA o faz em uma representação linear. Além disso, o método MLS$\bar{a}$ não faz aprendizado nem retrocesso não cronológico.

Entretanto, o método MLS$\bar{a}$ apresenta um desempenho idêntico ou melhor que o método MA para todas as instâncias experimentadas. É possível que, embora tenha tamanho quadrático no tamanho do grafo original, a fórmula *hard* utilizada pelo resolvedor apresentado contém propriedades que antecipam retrocessos cronológicos no processo de busca, não detectados pelo módulo de análise.

Também observa-se que o método MLS$\bar{a}$ tem desempenho superior ao MLS para quatro das instâncias experimentadas. Nestes casos, as cláusulas aprendidas pelo LIAMFSAT não têm impacto suficiente no processo de busca para torná-lo eficiente. Assim, manter a estrutura do Módulo de Inferências durante o processo não é relevante.

Por fim, é possível notar que o desempenho do método MLSj não é pior que o desempenho do método MLS, para a maioria das instâncias experimentadas. Este fato indica que, embora possa ser computacionalmente cara, a operação de subjugação não torna o processo do resolvedor mais lento, para as instâncias experimentadas.

Embora superior para a maioria das instâncias utilizadas, entretanto, o desempenho do método MLSj não apresenta diferença significativa (mais de 3 segundos) com o do método MLS. Assim, pode-se concluir que a melhoria proposta pela função de subjugação do Módulo de Inferência, para essas instâncias, embora não seja ruim na prática, não é suficiente para compensar significativamente seu *overhead*.

6.2 Ciclo Hamiltoniano

Esta seção apresenta os experimentos realizados com o problema de decisão do Ciclo Hamiltoniano. Além das reduções apresentadas nas seções 3.1 e 3.2, este problema em particular também foi experimentado com um método alternativo de resolução, que faz uso de resolvedores SAT. A próxima subseção apresenta este método, enquanto a última desta seção apresenta os resultados dos experimentos realizados.

6.2.1 Resolução do problema de decisão com SAT clausal

Além de reduções diretas, é possível resolver o problema do Ciclo Hamiltoniano com a utilização de resolvedores SAT através de outros métodos, como a verificação de existência de *modelos estáveis* [34].

É possível descrever uma instância de um problema de decisão como o do Ciclo Hamiltoniano como um programa em *Prolog*, cujo resultado é verdadeiro se e somente se a instância é positiva (o grafo é Hamiltoniano).

Ao invés de executar o programa em maquinários como o *Prolog*, é possível converter o programa para uma fórmula em CNF e fornecê-la como entrada para um resolvedor SAT clausal. Caso esta fórmula seja insatisfatível, o programa original certamente tem o valor falso como resultado.

Se a fórmula possui um modelo, entretanto, é necessário verificar se o mesmo é estável, isto é, se tal modelo indica que o programa original resultaria no valor verdadeiro. Em caso negativo, a fórmula é modificada e o resolvedor SAT é utilizado novamente. Este processo se repete até que um modelo estável seja encontrado ou até que a fórmula se torne insastisfatível [34].

O programa em *Prolog* utilizado nos experimentos para descrever o problema do Ciclo Hamiltoniano foi publicado anteriormente [39].

Além disso, o resolvedor *SUP* [34] foi utilizado como procedimento principal, que, por sua vez, utiliza o resolvedor SAT *Minisat* a cada iteração.

6.2.2 Resultados Obtidos

Os experimentos com este problema foram realizados com instâncias do *benchmark* do ASSAT para Ciclo Hamiltoniano [50], em particular as criadas "à mão" (*hand coded*).

A tabela 6.5 apresenta os tamanhos das fórmulas obtidas pelas reduções, enquanto a 6.6 apresenta o desempenho dos métodos para resolvê-las. A sigla CNFd indica a redução apresentada na subseção 3.1.2, e a resolução de suas instâncias pelo resolvedor SAT Lingueling.

Na tabela 6.6, a coluna *H?* indica se o grafo correspondente é Hamiltoniano (S) ou não (N). A sigla SUP se refere à técnica descrita na subseção 6.2.1.

Tabela 6.5: Tamanhos das fórmulas obtidas pelas reduções com instâncias criadas "à mão" do *Benchmark* do ASSAT

Instância	$\|V\|/\|E\|$	|CNFd|	|CNFr|
2xp30	60/158	3600/299700	33960/159658
2xp30.1	60/160	3600/299460	35280/164578
2xp30.2	60/160	3600/299460	34680/162838
2xp30.3	60/160	3600/299460	34680/162838
2xp30.4	60/159	3600/299580	34020/160558
4xp20	80/196	6400/727120	57920/261678
4xp20.1	80/199	6400/726640	58640/266958
4xp20.2	80/200	6400/726480	58880/268718
4xp20.3	80/200	6400/726480	58880/268718

Tabela 6.6: Resultados experimentais com instâncias criadas "à mão" do *Benchmark* do ASSAT

Instância	$\|V\|/\|E\|$	H?	SUP	CNFd	MLS	MLSj	MLS $\bar{a}$	MLS$\bar{m}$	MLS$\bar{a}\bar{m}$	MA	CNFr
2xp30	60/158	N	**0.1**	TLE	TLE	TLE	TLE	TLE	TLE	TLE	10.1
2xp30.1	60/160	S	**0.4**	TLE	TLE	TLE	TLE	TLE	TLE	TLE	2.9
2xp30.2	60/160	S	**1.3**	1188.3	TLE	TLE	TLE	TLE	TLE	TLE	2.4
2xp30.3	60/160	S	**1.3**	1192.4	TLE	TLE	TLE	TLE	TLE	TLE	2.4
2xp30.4	60/159	N	25.1	TLE	TLE	TLE	TLE	TLE	TLE	TLE	**20.8**
4xp20	80/196	N	**0.2**	TLE	TLE	TLE	TLE	TLE	TLE	TLE	4.0
4xp20.1	80/199	N	**1.5**	TLE	TLE	TLE	TLE	TLE	TLE	TLE	5.0
4xp20.2	80/200	S	5.2	TLE	TLE	TLE	TLE	TLE	TLE	TLE	**4.2**
4xp20.3	80/200	N	**1.2**	TLE	TLE	TLE	TLE	TLE	TLE	TLE	4.8

Como sugerido pelos autores da redução CNFd, também foram experimentadas ins-

tâncias compostas por grafos completos, e instâncias compostas por grafos completos adicionados de um vértice sem vizinhança.

As tabelas 6.7 e 6.8 apresentam os tamanhos e os desempenhos obtidos com essas instâncias, respectivamente. A instância Nk indica um grafo completo de N vértices, enquanto a instância Ng indica o mesmo grafo, mas com um vértice sem vizinhança a mais.

Tabela 6.7: Tamanhos das fórmulas obtidas a partir de grafos completos

Instância	$\|V\|/\|E\|$	|CNFd|	|CNFr|
9k	9/36	81/333	1458/6829
9g	10/36	100/640	1620/7588
10k	10/45	100/460	1850/9178
10g	11/45	121/836	2035/10096
11k	11/55	121/616	2541/12857
11g	11/45	144/1068	2772/14026

Tabela 6.8: Resultados experimentais com instâncias com grafos completos

Instância	$\|V\|/\|E\|$	H?	SUP	CNFd	MLS	MLSj	MLS $\bar{a}$	MLS$\bar{m}$	MLS$\bar{a}\bar{m}$	MA	CNFr
9k	9/36	S	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
9g	10/36	N	0.1	2.15	0.1	0.1	0.1	0.1	0.1	34.1	0.1
10k	10/45	S	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
10g	11/45	N	0.1	TLE	0.1	0.1	0.1	0.1	0.1	694.7	0.1
11k	11/55	S	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.1	0.2
11g	11/45	N	0.1	TLE	0.1	0.1	0.1	0.1	0.1	TLE	0.1
20k	20/190	S	0.3	0.1	TLE	TLE	TLE	TLE	TLE	0.1	1.3
20g	21/190	N	0.3	TLE	0.1	0.1	0.1	0.1	0.1	TLE	0.1

Através da análise das tabelas 6.5 e 6.7, é possível observar que a redução descrita na subseção 3.1.2 gera fórmulas menores do que a redução apresentada na subseção 3.2.5, quando convertidas para CNF. Isto também pode ser justificado por seus tamanhos assintóticos: A redução CNFd gera fórmulas que utilizam $|V|^2$ variáveis, enquanto as fórmulas provenientes da CNFr utilizam $O(|V||E|)$ variáveis. Em todas as instâncias experimentadas, $|V| < |E|$ e, em grafos completos, $|E| = \binom{|V|}{2}$. Assim, para grafos completos, a redução CNFr gera fórmulas com um número de variáveis cúbico no número de vértices do grafo, enquanto a CNFd gera fórmulas com um número quadrático.

Entretanto, pela análise das tabelas 6.6 e 6.8, é possível notar que as instâncias obtidas da redução CNFr são resolvidas mais rapidamente que as obtidas da redução CNFd. Todas as instâncias experimentadas do *Banchmark* foram resolvidas em tempo satisfatório (em até 30 segundos) pelo método CNFr, mas apenas duas foram resolvidas dentro do tempo limite pelo método CNFd. Isto indica que, mesmo maiores, as fórmulas obtidas da redução apresentada na subseção 3.2.5 são melhores que as obtidas pela redução da subseção 3.1.2.

Os experimentos realizados não apontaram uma diferença significativa entre os métodos que utilizam o resolvedor descrito no capítulo 5. Tal resolvedor não é capaz de resolver as instâncias do *Benchmark* no tempo estipulado, e resolve as instâncias com grafos completos rapidamente.

Observa-se, pela tabela 6.6, que o método CNFr tende a resolver as instâncias de grafos Hamiltonianos mais rapidamente que as de grafos não Hamiltonianos, de tamanho semelhante. Isto pode ser justificado pelo fato de que há várias valorações que satisfazem a fórmula gerada a partir de um grafo Hamiltoniano, e o resolvedor SAT pára quando encontra alguma. Como não há valoração que satisfaz a fórmula para as outras instâncias, o resolvedor SAT potencialmente explora o espaço de busca de todas as valorações, embora com cortes.

Pela análise da tabela 6.8, é possível notar também que o método MA resolve a instancia Nk muito mais rapidamente que a instância Ng, para todos os valores de N experimentados. Isto pode ser justificado pelo fato de que, como o Módulo de Análise corta as valorações parciais que representam grafos com ciclos pequenos, uma valoração que contém um ciclo hamiltoniano em um grafo completo é rapidamente encontrada. Enquanto isso, todas as valorações encontradas durante a resolução da instância Ng são invalidadas, devido ao vértice isolado adicionado.

Também é válido observar que todas as instâncias Ng foram resolvidas em tempo desprezível (menos de 1 segundo) pelo resolvedor apresentado no capítulo 5. Isto se deve ao fato de que a fórmula contém a restrição $C_1()$ (o operador Escolhe-1 com aridade zero), devido ao vértice sem vizinhança. Esta restrição é trivialmente insatisfatível, o que permite ao resolvedor concluir que não existem modelos para a fórmula de maneira

imediata.

É possível também observar que o comportamento do método MLSj não difere do apresentado na seção 6.1, por apresentar desempenho semelhante ao método MLS para todas as instâncias experimentadas.

6.3 Ciclo Hamiltoniano Mínimo

Esta seção apresenta os resultados obtidos dos experimentos realizados com instâncias do problema do Ciclo Hamiltoniano Mínimo. Foram experimentadas instâncias que são compostas por grafos completos, cujas arestas foram ponderadas aleatória e uniformemente.

Os métodos utilizados para Ciclo Hamiltoniano foram adaptados para os experimentos do problema de Ciclo Hamiltoniano Mínimo. No método CNFd em particular, a fórmula apresentada na subseção 3.1.2 foi utilizada como fórmula *hard*, enquanto subfórmulas que representam a ausência de cada aresta, convertidas para CNF, foram utilizadas como fórmulas *soft*.

As tabelas 6.9 e 6.10 apresentam os tamanhos e o desempenho dos métodos experimentos para este problema, respectivamente.

Tabela 6.9: Tamanhos das fórmulas obtidas a partir de grafos completos aleatórios para Ciclo Hamiltoniano Mínimo

Instância	$\|V\|/\|E\|$	|CNFd|	|CNFr|
5kw	5/10	145/495	200/823
6kw	6/15	246/876	306/1453
7kw	7/21	385/1414	539/2686
8kw	8/28	568/2136	736/3986
9kw	9/36	801/3069	1458/6865
10kw	10/45	1090/4240	1850/9223

Através da análise da tabela 6.9, é possível concluir que as instâncias geradas pela redução CNFd são menores que as geradas pela redução apresentada neste trabalho convertidas. Este fato também é observado no resultado dos experimentos para Ciclo Hamiltoniano.

Tabela 6.10: Resultados experimentais com grafos completos aleatórios para Ciclo Hamiltoniano Mínimo

Instância	$\|V\|/\|E\|$	CNFd	MLS	MLSj	MLS $\bar{a}$	MLS$\bar{m}$	MLS$\bar{a}\bar{m}$	MA	CNFr
5kw	5/10	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
6kw	6/15	0.1	1.4	1.5	5.0	2.2	6.0	0.1	0.1
7kw	7/21	0.1	68.2	66.3	205.8	134.8	284.2	0.1	0.1
8kw	8/28	4.6	TLE	TLE	TLE	TLE	TLE	0.4	**0.2**
9kw	9/36	14.4	TLE	TLE	TLE	TLE	TLE	1.1	**0.5**
10kw	10/45	308.0	TLE	TLE	TLE	TLE	TLE	17.8	**3.9**

Da mesma forma, as instâncias convertidas da redução apresentadas na subseção 3.2.6 são mais fáceis que as obtidas pela redução CNFd, como observado na tabela 6.10.

O método MA se apresentou mais eficaz do que o método MLS e suas variações nestes experimentos. Isto mostra que, para as instâncias experimentadas, a redução quadrática, o LIAMFSAT e o Módulo de Inferência são ineficazes ao lado de enumerador simples, que apenas corta os espaços de busca que contém ciclos não Hamiltonianos.

Esta observação é oposta à observada na tabela 6.8, que indica que o uso da redução quadrática apresentada e do LIAMFSAT é mais eficaz para o problema do Ciclo Hamiltoniano do que o método MA. Este contraste pode ser justificado pela natureza de ambos os problemas. Como o problema do Ciclo Hamiltoniano é de decisão, o processo do LIAMFSAT é finalizado assim que um modelo é encontrado, o que é rápido para grafos completos (tabela 6.8). Para o problema do Ciclo Hamiltoniano Mínimo, entretanto, tal modelo apenas atualiza a Cota Superior, e a busca prossegue até que todo o espaço seja exaurido.

Além disso, observa-se, pela análise da tabela 6.10, que ambos os módulos de Inferência e Análise melhoram o desempenho do resolvedor LIAMFSAT, para as instâncias experimentadas. O método MLS, para as instâncias que resolveu, teve um desempenho melhor que MLS$\bar{a}$, MLS$\bar{m}$ e principalmente MLS$\bar{a}\bar{m}$. Entretanto, ele não foi capaz de resolver instâncias de grafos com 8 vértices ou mais, consideradas pequenas para o problema, devido provavelmente à complexidade quadrática da redução utilizada.

O método MLSj também apresenta desempenho semelhante ao do método MLS, como já observado nas seções anteriores.

6.4 Clique

Esta seção apresenta os resultados dos experimentos realizados para o problema de Clique.

Os experimentos foram realizados com a redução para SAT apresentada na subseção 3.2.7 e com a redução conhecida do problema de Clique Máxima para MaxSAT, apresentada na subseção 3.1.3 (LCOMP).

Nos experimentos para este problema, foram utilizados grafos aleatórios de 1000 vértices cada, com densidade ($|E|/\binom{|V|}{2}$) entre 0.01 e 0.05. Utilizar grafos grandes e esparsos faz com que o tamanho das fórmulas geradas pela redução LCOMP, apresentada na subseção anterior, seja alto.

A tabela 6.11 apresenta os tamanhos das fórmulas CNF obtidas da redução LCOMP e da redução apresentada na subseção 3.2.7, convertidas. A coluna $|V|/|E|/maxk$ indica o número de vértices e arestas do grafo, além do tamanho de sua clique máxima. A coluna k indica o valor do parâmetro k utilizado nas reduções do problema de decisão para SAT.

Tabela 6.11: Tamanhos das fórmulas obtidas das reduções para (Max)SAT com grafos esparsos

Instância	$\|V\|/\|E\|/maxk$	|LCOMP|	k	|CNFr|
n1000d1	1000/4995/4	1000/495505	2	26401/140867
			3	26401/140866
			4	26399/140857
n1000d3	1000/14985/4	1000/485515	2	74685/425322
			3	74685/425322
			4	74685/425322
			5	74685/425322
n1000d5	1000/24975/5	1000/475525	2	123569/717487
			3	123569/717487
			4	123569/717487
			5	123569/717487
			6	123569/717487

Pela análise da tabela 6.11, é possível observar que a redução apresentada na subseção 3.2.7 gera instâncias que, quando convertidas para CNF, utilizam mais variáveis do que as geradas pela LCOMP. Isto é justificado pelo fato de que aquela redução gera fórmulas com $O(|E|)$ variáveis, enquanto a última gera fórmulas com $|V|$ variáveis, e $|E| > |V|$ nas

Tabela 6.12: Resultados obtidos com instâncias contendo grafos esparsos

Instância	$\|V\|/\|E\|/maxk$	LCOMP	k	MLS$\bar{m}$	MLSj$\bar{m}$	MLS$\bar{a}\bar{m}$	CNFr
n1000d1	1000/4995/4	0.6	2	0.2	0.2	0.2	5.8
			3	5.1	4.9	4.8	8.0
			4	TLE	TLE	TLE	338.5
n1000d3	1000/14985/4	1.1	2	TLE	TLE	TLE	23.5
			3	TLE	TLE	TLE	29.6
			4	TLE	TLE	TLE	TLE
			5	TLE	TLE	TLE	TLE
n1000d5	1000/24975/5	1.7	2	TLE	TLE	TLE	39.7
			3	TLE	TLE	TLE	43.7
			4	TLE	TLE	TLE	787.8
			5	TLE	TLE	TLE	TLE
			6	TLE	TLE	TLE	TLE

instâncias geradas.

Entretanto, nota-se que número de cláusulas, para instâncias de grafos suficientemente esparsos, é menor nas instâncias obtidas pela CNFr do que nas obtidas pela LCOMP. Isto ocorre para as instâncias com grafos de densidade 0.01 e 0.03. As fórmulas geradas pela CNFr para o grafo de densidade 0.05, entretanto, são maiores tanto em número de variáveis quanto de cláusulas do que as geradas pela redução LCOMP.

Pela análise da tabela 6.12, observa-se que o resolvedor Lingueling resolve as instâncias obtidas pela redução CNFr em tempo satisfatório quando as mesmas são satisfatíveis e o valor de k é pequeno, e não consegue resolver instâncias no tempo limite estipulado caso contrário. Para valores de k pequenos (como 2 ou 3), espera-se que um grafo aleatório possua mais de uma clique de tamanho k, o que indiretamente justifica tal comportamento do resolvedor SAT.

Também é possível observar, infelizmente, que o resolvedor apresentado no capítulo 5 não é capaz de resolver nenhuma instância relevante, dentre as experimentadas. Em particular, o resolvedor não resolve instâncias com $k = 2$, que consistem na trivial verificação de existência de alguma aresta em um grafo. É possível que a ausência de uma heurística de ramificação específica para o problema implique em tal desempenho, pois uma heurística que valore uma variável associada a qualquer aresta como verdadeiro e as demais como falso, nesta ordem, deve resolver instâncias com $k = 2$ rapidamente.

Por fim, o método LCOMP resolve um problema de otimização muito mais rapidamente que sua versão de decisão, para todas as instâncias experimentadas.

6.5 Considerações

Este capítulo apresenta os resultados obtidos com experimentos realizados. As principais observações realizadas neste capítulo são as de que as fórmulas geradas pelas novas reduções apresentadas no capítulo 3 são melhores do que as geradas por reduções conhecidas, mesmo quando seus tamanhos são maiores que as outras. A única exceção são as geradas pela redução de Clique.

Entretanto, uma melhoria significativa é obtida apenas quando as fórmulas geradas são convertidas para CNF, na maioria dos casos. O resolvedor apresentado no capítulo 5 não é mais eficiente que um resolvedor clausal no estado-da-arte, resolvendo instâncias convertidas com somadores e comparadores.

Os módulos adicionados a este resolvedor, entretanto, geralmente têm um impacto significativo no desempenho do mesmo. A função de subjugação do Módulo de Inferência, por sua vez, não demostrou tal impacto nos experimentos realizados.

O capítulo 7 apresenta uma conclusão sobre todo o trabalho e indica possíveis trabalhos futuros.

CAPÍTULO 7

CONCLUSÃO E TRABALHOS FUTUROS

Como apresentado, este trabalho tem dois principais objetivos: apresentar reduções de problemas em grafos cuja solução deve ser um subgrafo conexo para SAT e MaxSAT, e apresentar um resolvedor não clausal modificado capaz de resolver as instâncias geradas pelas reduções. Foram particularmente reduzidos neste trabalho os problemas da Árvore de Steiner, do Ciclo Hamiltoniano, do Ciclo Hamiltoniano Mínimo e de Clique.

As novas reduções apresentadas são lineares ou quadráticas no tamanho do grafo dado. Estes tamanhos são assintoticamente semelhantes ou melhores do que reduções previamente publicadas para os mesmos problemas, como observado.

Na prática, quando convertidas para CNF, a nova redução do problema da Árvore de Steiner gera fórmulas menores do que as geradas por outras reduções. Isto não é verdade, porém, para as novas reduções apresentadas para os os problemas de Ciclo Hamiltoniano e Mínimo, e Clique.

Entretanto, as fórmulas em CNF obtidas das novas reduções são resolvidas mais rapidamente por um resolvedor SAT no estado-da-arte do que as instâncias obtidas por outras reduções, mesmo para os problemas de Ciclo Hamiltoniano e Mínimo. Assim, pode-se concluir que as novas reduções, à única exceção da redução do problema de Clique, são todas melhores que reduções publicadas anteriormente, quando resolvidas por um resolvedor SAT eficiente.

O desempenho ruim da nova redução do problema de Clique permite também concluir o fato de que é possível que a codificação de restrições informais em fórmulas pode gerar instâncias mais fáceis que outras codificações, de mesma natureza. A restrição *"a solução deve conter um subgrafo completo"* utilizada na redução LCOMP, que pode ser vista como uma "especialização" da restrição *"a solução deve conter um subgrafo conexo"* utilizada na nova redução, é codificada de forma mais simples e gera fórmulas que são resolvidas mais

rapidamente. Assim, é computacionalmente melhor codificar uma dada característica (como *completo*) do que uma de suas subcaracterísticas (como *conexo*).

Embora melhores do que as geradas por reduções publicadas anteriormente em sua maioria, as instâncias obtidas pelas novas reduções não são boas o suficiente para fazer um resolvedor SAT no estado-da-arte conseguir resolver instâncias relevantes de problemas em grafos. Apenas algumas das menores instâncias do *benchmark* SteinLib, por exemplo, foram resolvidas no tempo limite estipulado. Essas e várias outras instâncias podem ser resolvidas em menos de um minuto por um computador atual com um método próprio para o problema [31].

Isto leva à conclusão esperada de que a redutibilidade entre problemas nem sempre deve ser utilizada para a resolução de instâncias relevantes na prática. Se um problema P_1 é reduzido a outro problema P_2, um dado algoritmo para P_2 possivelmente pode ser adaptado para a resolução de P_1, o que inutiliza a redução neste caso. Se a redução não é linear, é esperado que o método utilizado para a resolução de P_2 seja pior que sua adaptação para P_1, por trabalhar em uma representação não linear da instância original.

Além de resolvedores clausais, foi experimentado um resolvedor SAT não clausal que suporta o operador Escolhe-H de forma direta, e que utiliza as técnicas de aprendizado, retrocesso não cronológico, inferência de literais e possivelmente subjugação, além do auxílio de um módulo que atua na representação original do grafo. Tais técnicas, mesmo quando aplicadas a uma representação de tamanho quadrático do grafo original, se mostraram melhores do que um método de retrocesso ou ramificação e poda que trabalha com a representação linear do mesmo, como notado pela comparação dos métodos MLS e MA no capítulo 6. Isto demostra que as técnicas utilizadas por resolvedores SAT no estado-da-arte são de fato impactantes no desempenho do mesmo.

Entretanto, mesmo utilizando essas novas técnicas, o LIAMFSAT é muito mais ineficiente do que um resolvedor SAT clausal no estado-da-arte, para a maioria das instâncias experimentadas. Resolvedores clausais tendem a ser mais eficientes do que os não clausais resolvendo a mesma instância, possivelmente devido ao grande esforço da comunidade aplicado nestas técnicas, ultimamente.

É válido, entretanto, reimplementar o operador Escolhe-H em um outro resolvedor SAT não clausal, e integrar tal resolvedor com o Módulo de Análise. Como trabalho futuro, pode-se modificar o NOCLAUSE de forma a torná-lo um resolvedor MaxSAT que suporte o operador. O NOCLAUSE utiliza técnicas diferentes que o LIAMFSAT tanto durante a propagação de literais quanto na inferência de valores verdade, como descrito. Embora talvez ainda não seja tão eficiente quanto um resolvedor clausal, tais modificações no NOCLAUSE podem resultar em um desempenho melhor ao obtido neste trabalho com o LIAMFSAT.

Também pode ser interessante, futuramente, utilizar uma abordagem diferente da ramificação e poda para resolver as instâncias de MaxSAT obtidas. Como apresentado no capítulo 4, há técnicas diferentes que podem ser utilizadas para resolver o problema. É possível que tais técnicas se mostrem melhores na resolução dessas instâncias específicas.

É interessante também estudar a operação de subjugação. A operação não é realizada nos resolvedores SAT no estado-da-arte, mas foi experimentada neste trabalho. Infelizmente, o desempenho do resolvedor não foi melhorado de maneira significativa pelo uso dessa operação, para as instâncias experimentadas. Entretanto, tal operação pode ter um impacto maior quando utilizada para resolver outras instâncias de SAT. É possível que haja fórmulas que "induzam" um resolvedor SAT a aprender uma determinada coleção de cláusulas, na qual várias delas subjugam várias outras. Neste caso, a operação de subjugação pode ser relevante.

Também como trabalho futuro, pode-se modificar um resolvedor SAT no estado-da-arte, como o Lingeling, para que a operação de subjugação descrita seja utilizada. Como apresentado, isto pode fazer com que o desempenho do resolvedor seja melhorado para um dado conjunto de instâncias.

BIBLIOGRAFIA

[1] Teresa Alsinet, Felip Manyà, e Jordi Planes. An efficient solver for weighted max-sat. *Journal of Global Optimization*, 41(1):61–73, 2008.

[2] Anbulagan e Alban Grastien. Importance of variables semantic in cnf encoding of cardinality constraints. *Symposium on Abstraction, Reformulation and Approximation (SARA'09)*, 2009.

[3] Carlos Ansótegui, María Luisa Bonet, e Jordi Levy. Solving (weighted) partial max-sat through satisfiability testing. volume 5584, páginas 427–440. Springer-Verlag, Springer-Verlag, 2009.

[4] Josep Argelich, Chu Min Li, Felip Manyà, e Jordi Planes. Max-sat 2011 - sixth max-sat evaluation, Modificado em Abril de 2011. http://www.maxsat.udl.cat/11/.

[5] Mohamed El bachir Menai. A logic-based approach to solve the steiner tree problem. *Artificial Intelligence Applications and Innovations III, Proceedings of the 5TH IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI 2009)*, volume 296 of *IFIP*, páginas 73–79. Springer, 2009.

[6] Olivier Bailleux. Boolvar/pb v1.0, a java library for translating pseudo-boolean constraints into cnf formulae. *Computing Research Repository (CoRR)*, abs/1103.3954, 2011.

[7] Olivier Bailleux e Yacine Boufkhad. Full cnf encoding: The counting constraints case. *in 7th Intl. Conf. on Theory and Applications of SAT Testing*, 2004.

[8] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, e Cesare Tinelli. Satisfiability modulo theories. Armin Biere, Marijn Heule, Hans van Maaren, e Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, páginas 825–885. IOS Press, 2009.

[9] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962.

[10] S. Bevc e I. Savnik. Using tries for subset and superset queries. *Information Technology Interfaces, 2009. ITI '09. Proceedings of the ITI 2009 31st International Conference on*, páginas 147 –152, Junho de 2009.

[11] A. Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. Relatório Técnico 10/1, Institute for Formal Models and Verification, Johannes Kepler University, 2010.

[12] M. Buro e H. K. Büning. Report on a sat competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, 1993.

[13] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, páginas 151–158, New York, NY, USA, 1971. ACM.

[14] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, e Charles E. Leiserson. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[15] Martin Davis, George Logemann, e Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, Julho de 1962.

[16] Ricardo Tavares de Oliveira, Fabiano Silva, Bruno Ribas, e Marcos Castilho. On modeling connectedness in reductions from graph problems to extended satisfiability. volume 7637 of *Lecture Notes in Computer Science*, páginas 442–451. Springer-Verlag, 2012.

[17] Emanuele Di Rosa, Enrico Giunchiglia, e Marco Maratea. Solving satisfiability problems with preferences. *Constraints*, 15:485–515, Outubro de 2010.

[18] Edsger Wybe Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerical Mathematics*, 1:269–271, 1959.

[19] Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24:327–336, 1994.

[20] Jon William Freeman. *Improvements To Propositional Satisfiability Search Algorithms*. Tese de Doutorado, University of Pennsylvania, 1995.

[21] Zhaohui Fu e Sharad Malik. On solving the partial max-sat problem. *In International Conference on Theory and Applications of Satisfiability Testing (SAT'06), LNCS 4121*, páginas 252–265. Springer, 2006.

[22] B. Fuchs, W. Kern, D. Mölle, S. Richter, P. Rossmanith, e X. Wang. Dynamic programming for minimum steiner trees. *Theory of Computing Systems*, 41:270–280, 2007.

[23] Federico Heras e Javier Larrosa. New inference rules for efficient max-sat solving. *Proceedings of the 21st national conference on Artificial intelligence*, 1:68–73, 2006. AAAI Press.

[24] Federico Heras, Javier Larrosa, e Albert Oliveras. Minimaxsat: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[25] Alexander Hertel, Philipp Hertel, e Alasdair Urquhart. Formalizing dangerous sat encodings. *Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, SAT'07, páginas 159–172. Springer-Verlag, 2007.

[26] Jinbo Huang. The effect of restarts on the efficiency of clause learning. *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, páginas 2318–2323, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[27] Himanshu Jain e Edmund M. Clarke. Efficient sat solving for non-clausal formulas using dpll, graphs, and watched cuts. *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, páginas 563–568, New York, NY, USA, 2009. ACM.

[28] R. G. Jeroslow e J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, (1):167–187, 1990.

[29] R. M. Karp. Reducibility Among Combinatorial Problems. R. E. Miller e J. W. Thatcher, editors, *Complexity of Computer Computations*, páginas 85–103. Plenum Press, 1972.

[30] Henry Kautz, Bart Selman, e Yueyen Jiang. A general stochastic approach to solving problems with hard and soft constraints. *The Satisfiability Problem: Theory and Applications*, páginas 573–586. American Mathematical Society, 1996.

[31] Thorsten Koch, Alexander Martin, e Stefan Voš. Steinlib: An updated library on steiner tree problems in graphs. Relatório Técnico 00-37, Zuse-Institut Berlin (ZIB), Novembro de 2000.

[32] Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, Fevereiro de 1956.

[33] Javier Larrosa, Federico Heras, e Simon de Givry. A logical approach to efficient max-sat solving. *ARTIFICIAL INTELLIGENCE*, 172:204–233, 2008. ACM.

[34] Yuliya Lierler. Abstract answer set solvers. *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, páginas 377–391. Springer Berlin Heidelberg, 2008.

[35] Joao Marques-Silva e Inês Lynce. Towards robust cnf encodings of cardinality constraints. *Proc. 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), Vol 4741 OF LNCS*, páginas 483–497. Springer, 2007.

[36] Joao P. Marques-Silva e Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. *in Proceedings of the International Conference on Computer-Aided Design*, páginas 220–227, 1996.

[37] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, e Sharad Malik. Chaff: Engineering an efficient sat solver. *ANNUAL ACM IEEE DESIGN AUTOMATION CONFERENCE*, páginas 530–535. ACM, 2001.

[38] D. E. Muller e F.P. Preparata. Bounds to complexities of networks for sorting and for switching. *Journal of the ACM*, 22:195 – 201, 1975.

[39] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[40] M. Pipponzi e F. Somenzi. An iterative algorithm for the binate covering problem. *Proceedings of the conference on European design automation*, EURO-DAC '90, páginas 208–211, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[41] Bruno César Ribas. Satisfabilidade não-clausal restrita às variáveis de entrada. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Brasil, 2011.

[42] Bart Selman, Hector Levesque, e David Mitchell. A new method for solving hard satisfiability problems. *National Conference on Articifial Intelligence*, páginas 440–446, 1992. AAAI Press.

[43] Kevin Smyth, Holger H. Hoos, e Thomas Stützle. Iterated robust tabu search for max-sat. *In Proc. of the 16th Conf. of the Canadian Society for Computational Studies of Intelligence*, 2671:129–144, 2003. Springer.

[44] Christian Thiffault, Fahiem Bacchus, e Toby Walsh. Solving non-clausal formulas with dpll search. *In SAT*, páginas 663–678. Springer, 2004.

[45] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Structures in Constructive Mathematics and Mathematical Logic*, 2:115–125, 1968.

[46] Richard Wallace e Eugene C. Freuder. Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. *Cliques, Coloring and Satisfiability*, páginas 587–615, 1996. American Mathematical Society.

[47] D. M. Warme, P. Winter, e M. Zachariasen. Exact algorithms for plane steiner tree problems: A computational study. *Advances in Steiner Trees*, páginas 81–116. Kluwer Academic Publishers, 1998.

[48] Hui Xu, R.A. Rutenbar, e K. Sakallah. sub-sat: a formulation for relaxed boolean satisfiability with applications in routing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(6):814 – 820, Junho de 2003.

[49] Hantao Zhang. Sato: An efficient propositional prover. *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, páginas 272–275, 1997.

[50] Yuting Zhao. Assat - hamiltonian circuit (hc) domain, Acessado em Outubro de 2011. http://assat.cs.ust.hk/Assat-2.0/hc-2.0.html.